I-8120W

User's Manual

Warranty

All products manufactured by ICP DAS are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

Warning

ICP DAS assume no liability for damages consequent to the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS for its use, nor for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

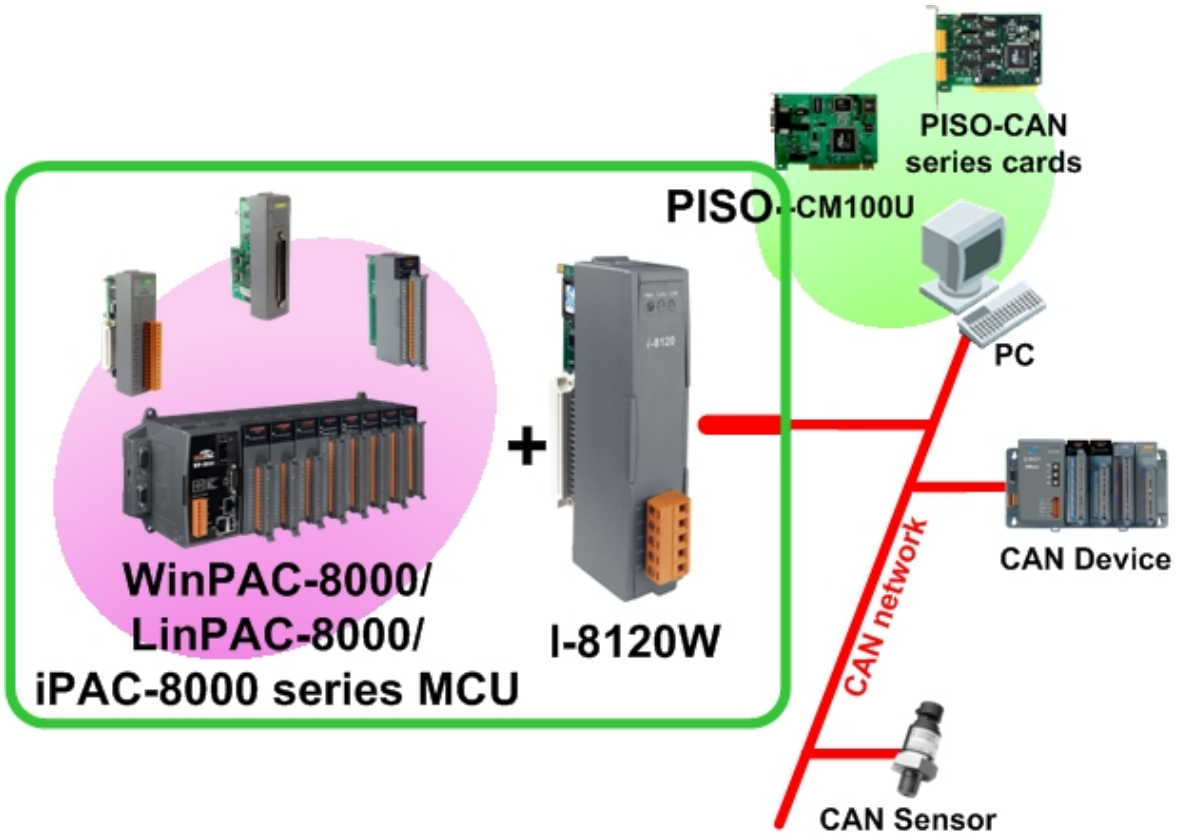
The names used for identification only maybe registered trademarks of their respective companies.

Tables of Content

1 General Information

1.1 Introduction

The CAN (Controller Area Network) is a serial communication bus especially suited to interconnect smart devices to build smart systems or sub-system. As standalone CAN controller, I-8120W with WinPAC, LinPAC and iPAC series MCU (main control unit) represents an economic solution. It has one CAN communication ports with 5-pin screw terminal connector, and is useful to a wide range of CAN applications. Besides, I-8120W uses the NXP SJA1000T and transceiver 82C250, which provide both CAN 2.0A and 2.0B specific, re-transmission function, bus arbitration and error detection. By owing to the benefits of WinPAC, LinPAC and iPAC series MCU without increasing the CPU loading heavily, it is a powerful one-CAN-port programmable device server by driving the CAN port of I-8120W with dual port RAM. Therefore, Users can combine the advantage of WinPAC, LinPAC and iPAC series MCU with I-8120W, and apply them on various industrial applications.



1.2 Features

- Support WinPAC-8000 series MCU (driver for LinPAC and iPAC series MCU will be available soon)
- Follow ISO11898-2 specification
- NXP SJA1000T CAN controller
- NXP 82C250 CAN transceiver
- CAN controller frequency :16 MHz
- 2500Vrms photo-isolation protection on CAN side
- Switch for 120Ω terminator resistor of CAN bus
- One CAN communication port
- Compatible with CAN specification 2.0 parts A and B
- Provide default baud rate: 10 kbps, 20 kbps, 50 kbps, 125 kbps, 250 kbps, 500 kbps, 800 kbps, and 1 Mbps
- Allow user-defined baud rate
- 2048 records reception buffer and 256 records transmission buffer
- Cyclic transmission precision: ±0.5ms precision when cyclic time is below 10ms , ±1% error when cyclic time exceeds 10ms.
- Provide 5 sets of cyclic transmission.
- Timestamp of CAN message with ±1ms precision
- 80186, 80 MHz CPU
- 8 Kbytes DPRAM inside
- RTC(Real Time Clock) inside
- Allow user to program user-defined firmware in I-8120W
- Firmware updatable
- eVC++ demos and libraries are given
- C/C++ function libraries of firmware is given

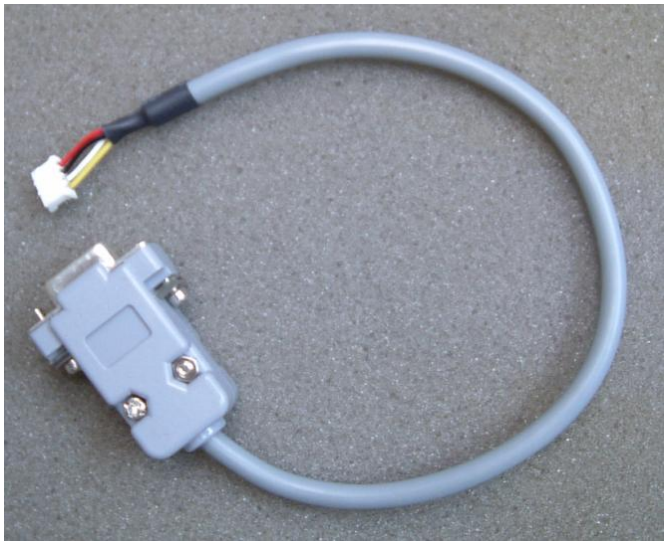
1.3 Specifications

- CAN controller: Phillips SJA1000T
- CAN controller frequency :16 MHz
- CAN transceiver: Phillips 82C250.
- Follow ISO11898-2 specification
- One CAN communication port
- Compatible with CAN specification 2.0 parts A and B
- Jumper select 120Ω terminator resistor for CAN bus
- Provide default baud rate: 10 kbps, 20 kbps, 50 kbps, 125 kbps, 250 kbps, 500 kbps, 800 kbps, and 1 Mbps
- Allow user-defined baud rate
- Connector: 5-pin screw terminal connector
- Isolation voltage: 2500Vrms on CAN side
- 80186, 80 MHz CPU
- 8 Kbytes DPRAM (1 Kbytes for system)
- 512 Kbytes Flash memory (128 Kbytes for system, others for firmware)
- 512 Kbytes SRAM
- RTC (real time clock) inside
- 2 Kbytes EEPROM (256 bytes for system)
- 31 bytes NVRAM
- 3 indication LED (Red LED for power, Yellow and Green LEDs for users)
- Power requirements: 2W
- Environment:
  Operating temp: -25°C ~ +75°C
  Storage temp: -30°C ~ +85°C
  Humidity: 5% ~ 95% RH non-condensing

1.4 Product Check List

Besides this manual, the package includes the following items:

- □ I-8120W CAN module
- □ Software CD ROM
- □ Release note and Quick start
- □ One debug cable (model number is 4PCA-0904)

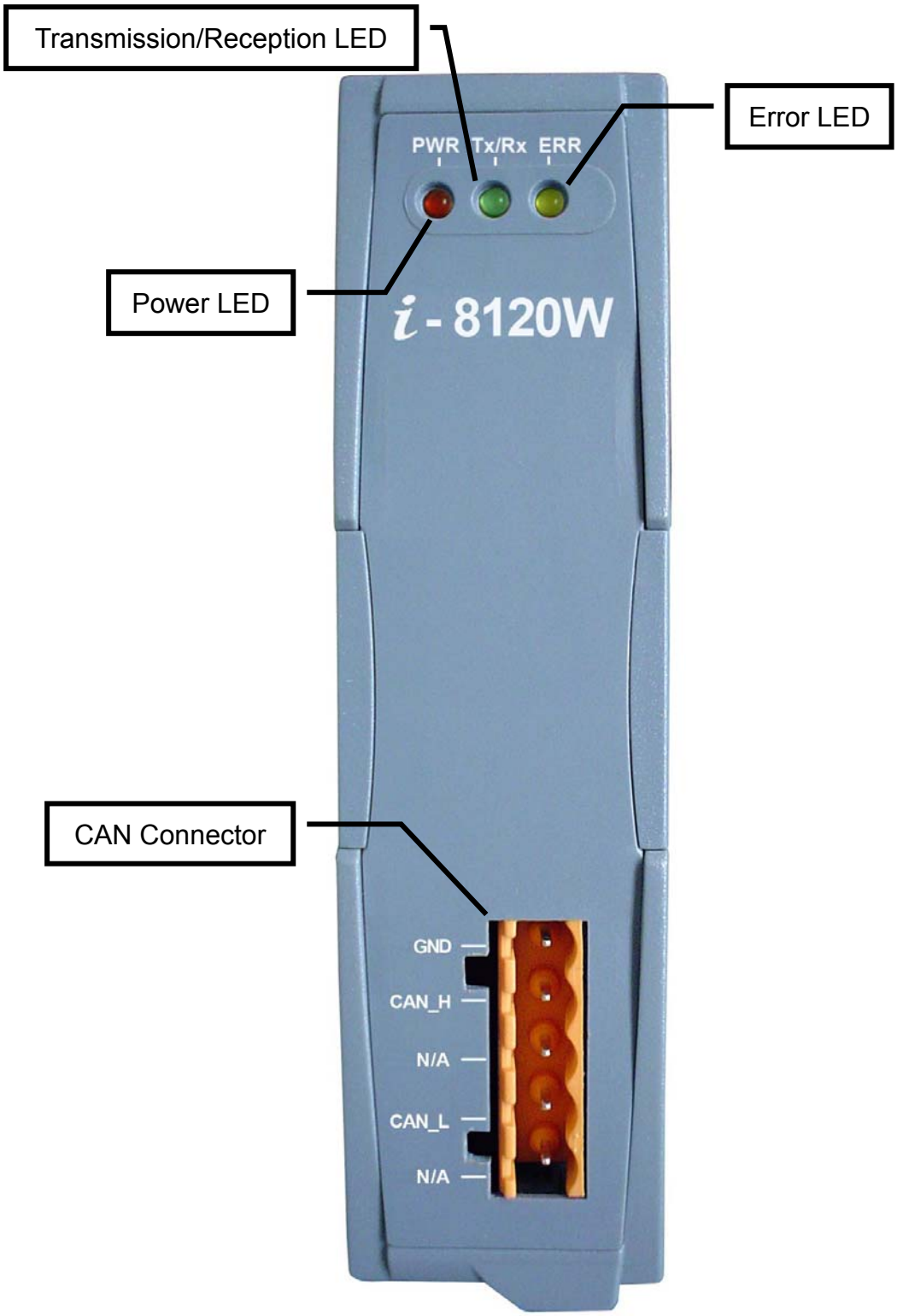


Attention !

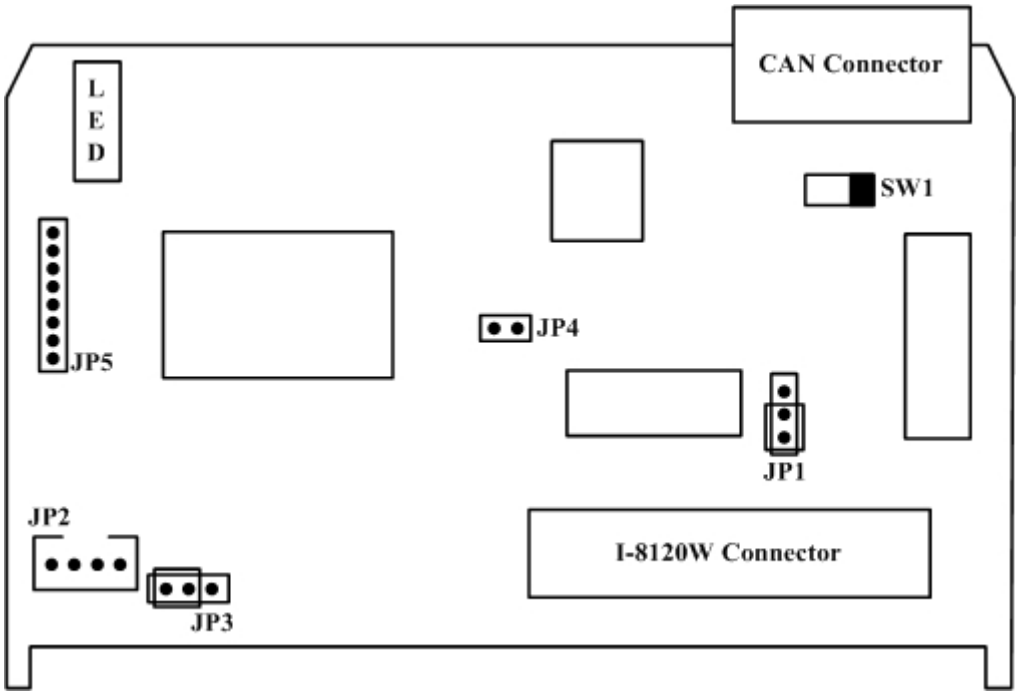
If any of these items are missing or damaged, please contact your local field agent. Keep aside the shipping materials and carton in case you want to ship or store the product in the future.

2 Hardware Configuration

2.1 Hardware Profile



2.2 Jumper Selection



The following table shows the definition of jumpers or switch. Users need to refer to this table to configure the I-8120W hardware.

Jumper	Description	Status	
SW1	CAN Port 120Ω terminal resistance.	SW1 Enable	SW1 Disable
JP1	Jumper for writing protection of flash memory. It can lock/unlock the flash memory to enable/disable the firmware download procedure. In harsh environment, it is useful to prevent flash memory access from noise or disturbances.	JP1 Lock	JP1 Unlock

Jumper	Description	Status	
JP2	Debug port for user-defined firmware. Users can connect the debug port with the PC RS-232 port via the debug cable.	 4-pin connector for JP2 D-Sub 9 pin connector for PC RS-232 port	
JP4	When users download the firmware which is not proper for I-8120W or has some bug so that the I-8120W can't work normally. Set this jumper to reset status until the green and yellow LEDs of I-8120W interlace flash once per second. Then, set this jumper to normal status. Afterwards, the I-8120W is in download mode. Users can kill the old firmware and download the new one into I-8120W by using Utility tool. If users want to run default firmware or user-defined firmware normally, please keep this jumper in normal status. Please section 2.5 for more detail about LED action and I-8120W mode.	 Reset	 Normal

Table 2.1 Jumper or switch selections

2.3 Connector Pin Assignment

The I-8120W is equipped with one **5-pin screw terminal connector** for wire connection of the CAN bus. The connector's pin assignment is specified as following:

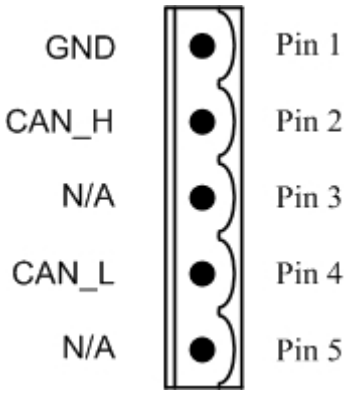


Figure2.2 5-pin screw terminal connector

Pin No.	Signal	Description
1	GND	Ground
2	CAN_H	CAN_H bus line (dominant high)
3	N/A	Non-available
4	CAN_L	CAN_L bus line (dominant low)
5	N/A	Non-available

Table 2.2: Pin assignment of 5-pin screw terminal connector

2.4 Wire connection

In order to minimize the reflection effects on the CAN bus line, the CAN bus line has to be terminated at both ends by two terminal resistances as in the following figure. According to the ISO 11898-2 spec, each terminal resistance is 120Ω (or between 108Ω~132Ω). The length related resistance should have 70 mΩ/m. Users should check the resistances of the CAN bus, before they install a new CAN network.

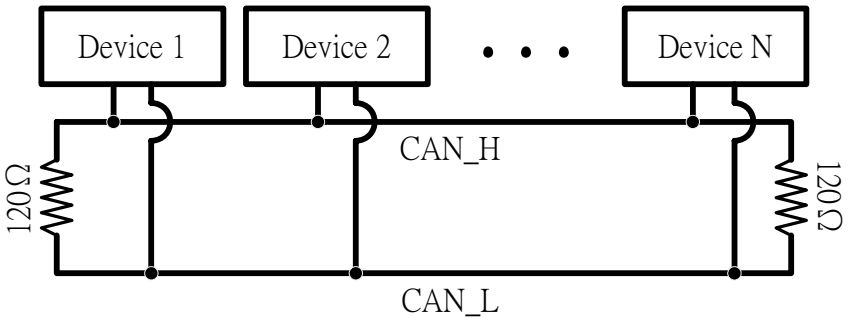


Figure 2.4 CAN bus network topology

Moreover, to minimize the voltage drop over long distances, the terminal resistance should be higher than the value defined in the ISO 11898-2. The following table can be used as a good reference.

Bus Length (meter)	Bus Cable Parameters		Terminal Resistance (Ω)
	Length Related Resistance (mΩ/m)	Cross Section (Type)	
0~40	70	0.25(23AWG)~ 0.34mm^2(22AWG)	124 (0.1%)
40~300	< 60	0.34(22AWG)~ 0.6mm^2(20AWG)	127 (0.1%)
300~600	< 40	0.5~0.6mm^2 (20AWG)	150~300
600~1K	< 20	0.75~0.8mm^2 (18AWG)	150~300

Table 2.4 Relationship between cable characteristics and terminal resistance

2.5 LED Indicator & I-8120W Mode

The LED status is changed when I-8120W is in different mode. There are three modes, and each mode describes as following:

1. Download mode: In this case, Green and yellow LEDs interlace flash once per second. At the same time, I-8120W is ready to update the firmware by Utility. Therefore, users can use Utility to download the newer default firmware or the user-defined firmware into I-8120W.

2. Firmware mode: If users use default firmware of I-8120W, the green LED will be flashed once when I-8120W receive/transmit one CAN message to CAN bus successfully. If bus loading is heavy, the green LED will turn on always. When some error occurs, the yellow LED will be turned on. Users can use I8120_Status() function to get the situation except buffer status. Reading or sending CAN messages can get the buffer status from the return code of functions. If I-8120W uses user-defined firmware, users can design the action of green LED or yellow LED by themselves.

3. Reset mode: If users reset the I-8120W by JP4 described in section 2.2, both green and yellow LED will turn on about 1 second. Afterwards, I-8120W is forced to enter the download mode. When I-8120W is out of control because the bug of user-defined firmware or other problems, use this method to reset firmware and download newer firmware again. Note that if users always set the JP4 to reset status, the I-8120W will switch the mode between reset mode and download mode.

2.6 Hardware Installation

When users want to use I-8120W, the hardware installation needs to be finished as following steps.

1. Shutdown your main control unit (WP-8xxx, LP-8xxx, or iPAC-8xxx).
2. Configure the SW1 for the terminal resistance. Check JP1 and JP4 status. If users want to update the firmware of I-8120W, set JP1 to unlock mode. The more detail information could be found on the session 2.2.
3. If users want to debug the firmware of I-8120W from debug port, connect the JP3 with special cable. For more detail, please refer to session 2.2.
4. Plug the CAN bus cable(s) into the 5-pin screw terminal connector.

When the procedure described above is completed, plug I-8120W into proper slot of main control unit, then turn on the main control unit.

3 I8120W Programming

In this chapter, it shows that how to develop the application and user-defined firmware. Section 3.1 describes the software architecture of I-8120W. Section 3.2 shows the procedures of programming an application by using default firmware. Some application demos are given here. Section 3.3 introduces how to build application according to your user-defined firmware by step-by-step method. Section 3.4 introduces the I8120W_Utility. When users want to update the default firmware or download user-defined firmware into I-8120W. This tool must be used. Section 3.5 shows the basic concept about the relationship between user-defined firmware and the corresponding application. Section 3.6 gives a profile about how to design the user-defined firmware by step-by-step introduction. Section 3.7 provides two ways to debug the user-defined firmware. If users just use the default firmware for their application, the Section 3.5, 3.6 and 3.7 can be ignored.

3.1 Software Architecture

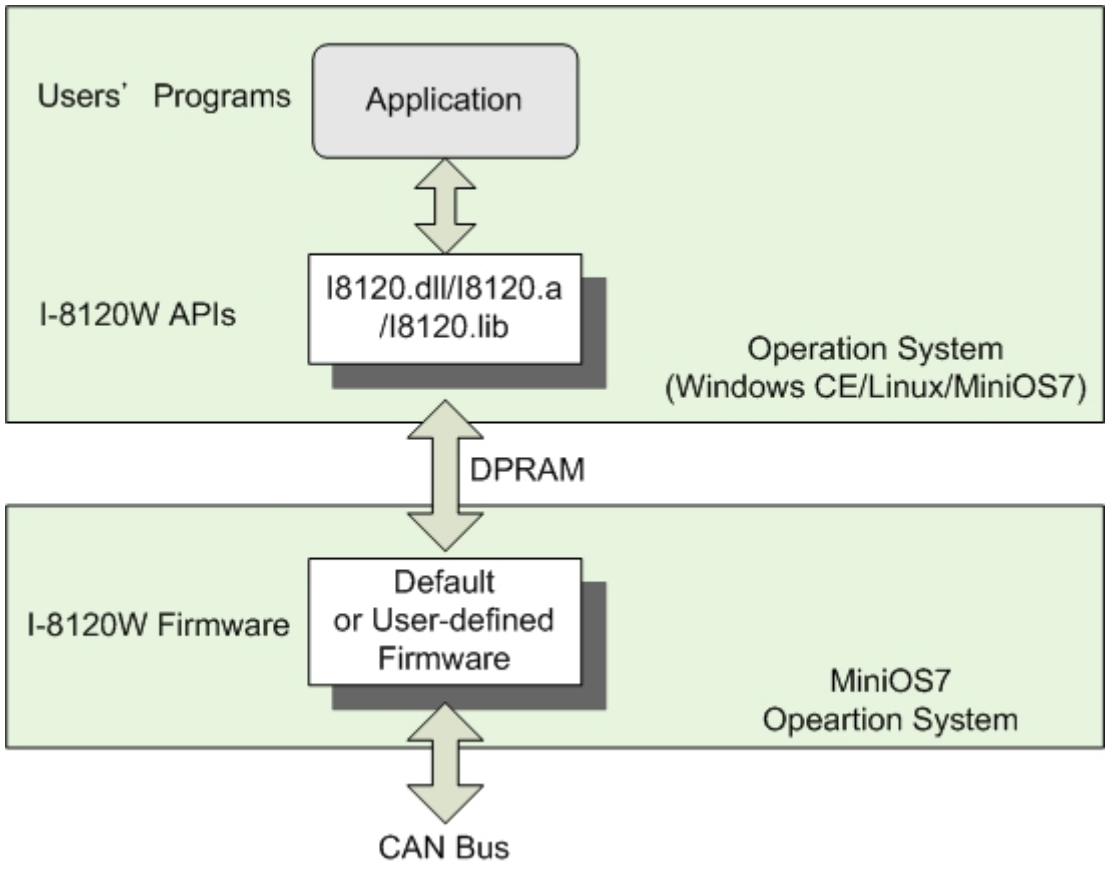


Figure 3.1 I-8120W Basic Software Architecture

The basic software architecture of I-8120W in WinPAC/LinPAC/iPAC is shown as figure 3.1. Take WinPAC for the example. The Windows CE APIs for I-8120W are provided by I8120.dll. Users can apply this dll file with embedded Visual C++ to create the WinCE applications. The Windows CE applications will communicate with I-8120W via DPRAM. Besides the basic functions for the general purpose applications, users can even design their special firmware for various CAN applications. If users just need the general functions, apply the APIs marked with "<for default firmware>" to build their Windows applications. These APIs provide users to configure the CAN controller, get the status of CAN controller, send/receive CAN messages to/from CAN bus and send CAN messages with cyclic transmission engine. These features help users to reach the purposes of bus monitor, bus access, network debugging, network setup … and etc. The software architecture is shown below.

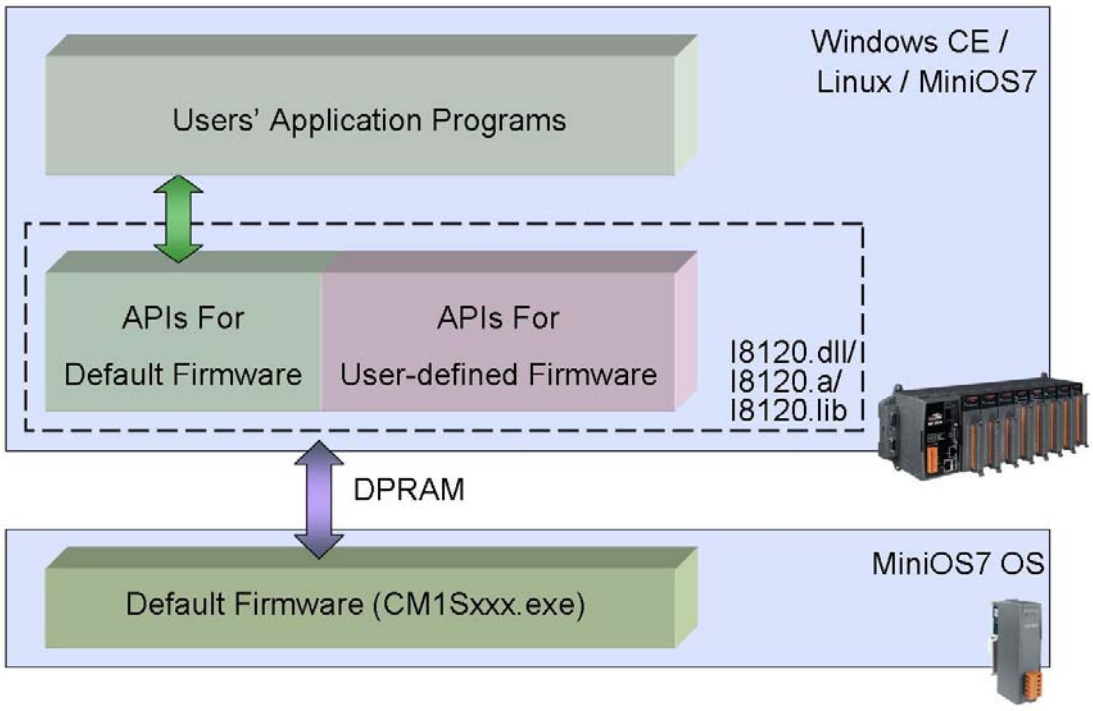


Figure 3.2 I-8120W Default Firmware Software Architecture

In some special cases, I-8120W provides the flexibilities to arrange the user-defined firmware. This feature may be helpful and powerful for some applications which have complex application protocols or need to improve the system efficiency. Users can interpret the raw CAN messages by the pre-defined application protocols on MiniOS7 platform, and feedback the useful and simplified data to users' Windows applications. This software architecture can

have the real-time processing feature, increase the execution performance and efficiently reduce the WP-8000 CPU loading. The software architecture is shown below.

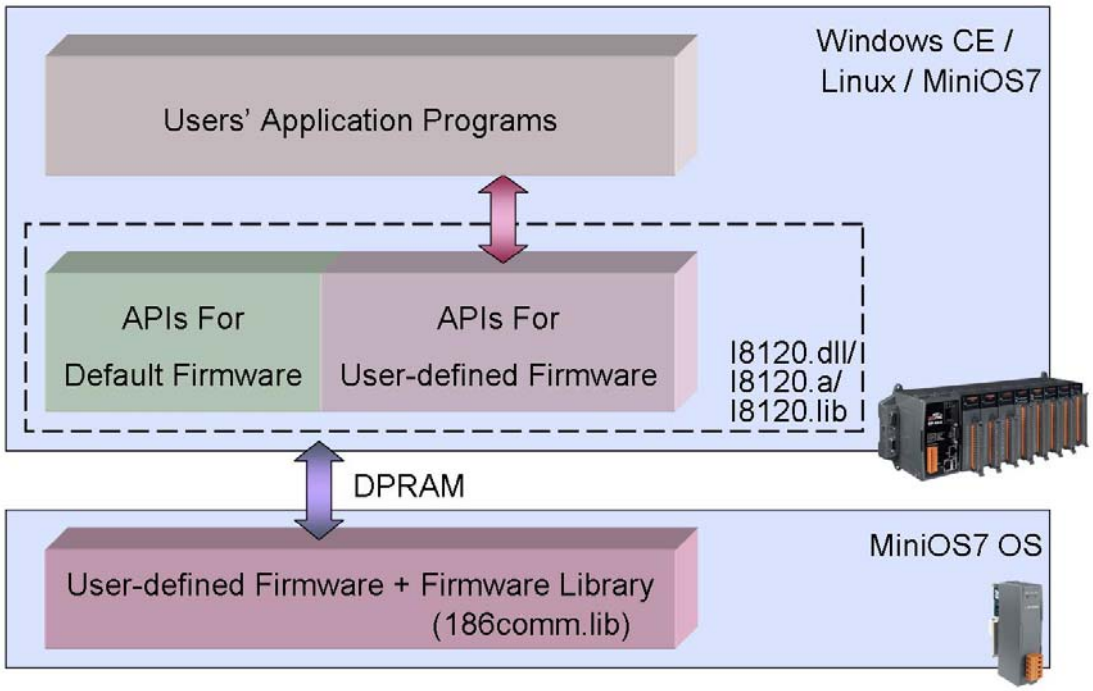


Figure 3.3 I-8120W User-defined Firmware Software Architecture

3.2 Application Programming With Default Firmware

This section is only for default firmware of I-8120W. It is useless if users want to develop their user-defined firmware of I-8120W. Figure 3.4 is a standard procedure for receiving a CAN messages. This procedure let users obtain the CAN messages from CAN bus easily. Figure 3.5 presents the “Send CAN Message” procedure. When users want to design their application by using the APIs of I8120.dll on WinPAC\LinPAC\iPAC, this flowchart may be a good reference. If users need to send some specified CAN messages every period of time, the flowchart shown in figure 3.6 may give a good example. Owing to these procedures, it may satisfy most application of users’ application with default firmware of I-8120W. Following the operation principle of I-8120W can help users with building their application easier and faster. When users want to combine these three procedure for various application, the functions, I8120_Init(), I8120_HardwareReset(), and I8120_Config(), are only called once during the start of application. Users can check the demo of default firmware to know how to implement these flowcharts on users’ application. The following list shows all of the demos of I-8120W for default firmware. These demos are only used when the default firmware is inside the I-8120W. Users can learn the basic skills about how to setup an application of I-8120W on WinPAC/LinPAC/iPAC main control unit.

```
|--\Demos                    → I-8120W demo programs
   |--\WinPAC_Lib            → The WinPAC APIs of I-8120W
   |--\For_Default_Firmware  → Folder for default firmware
      |--\Default_Firmware   → Default firmware of I-8120W
      |--\WinPAC             → Folder for WinPAC demo
         |--\eVC++           → Folder for the demos by eVC++
            |--\ RecMsg      → Demo for receiving CAN messages
            |--\ SendMsg     → Demo for sending CAN messages
            |--\ SendCyclicMsg → Demo for sending CAN messages cyclically
```

Note: In the following flowcharts, users must call I8120_RestoreI1820 function before user close the application on WinPAC/LinPAC/iPAC main control. Or, the interrupt and system resource will not be released, and it may cause system crash or make system be unstable.

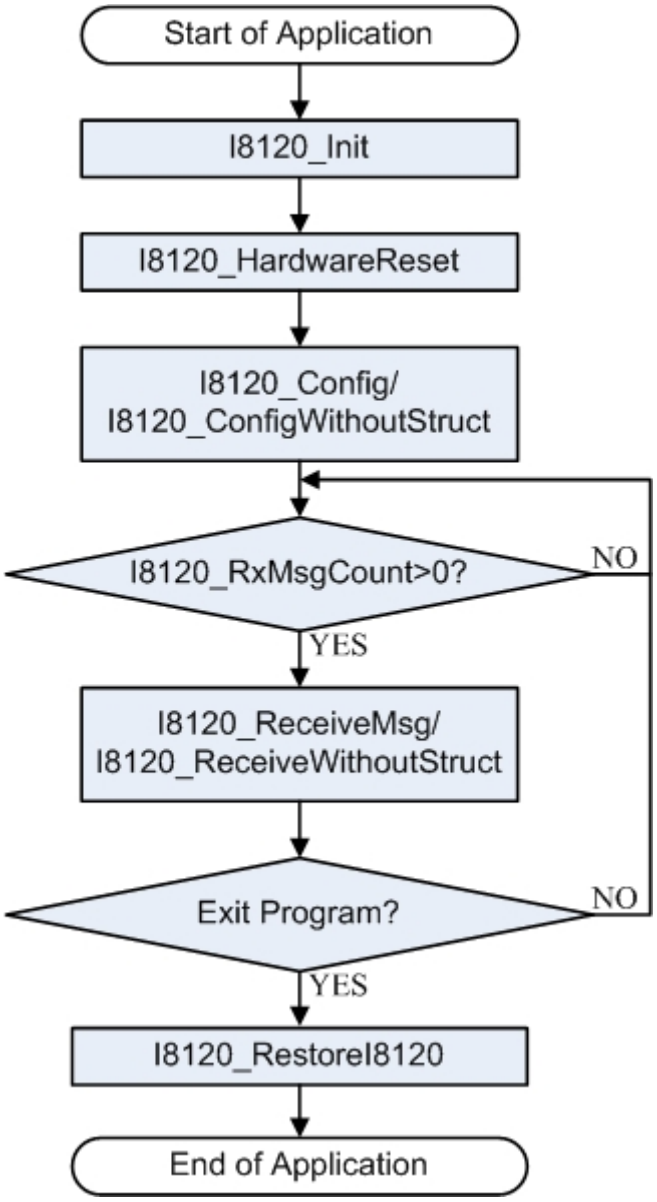


Figure 3.4 Flowchart of Receiving CAN Massages

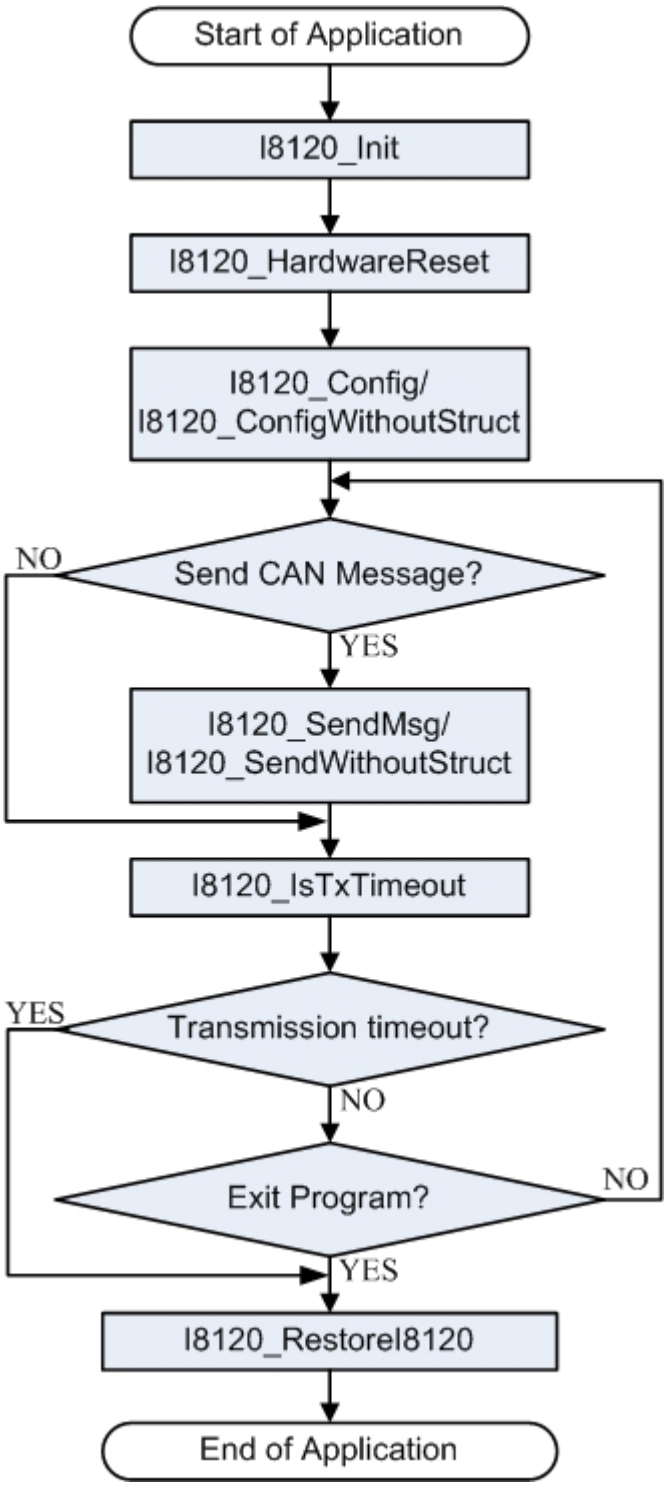


Figure 3.5 Flowchart of Sending CAN Massages

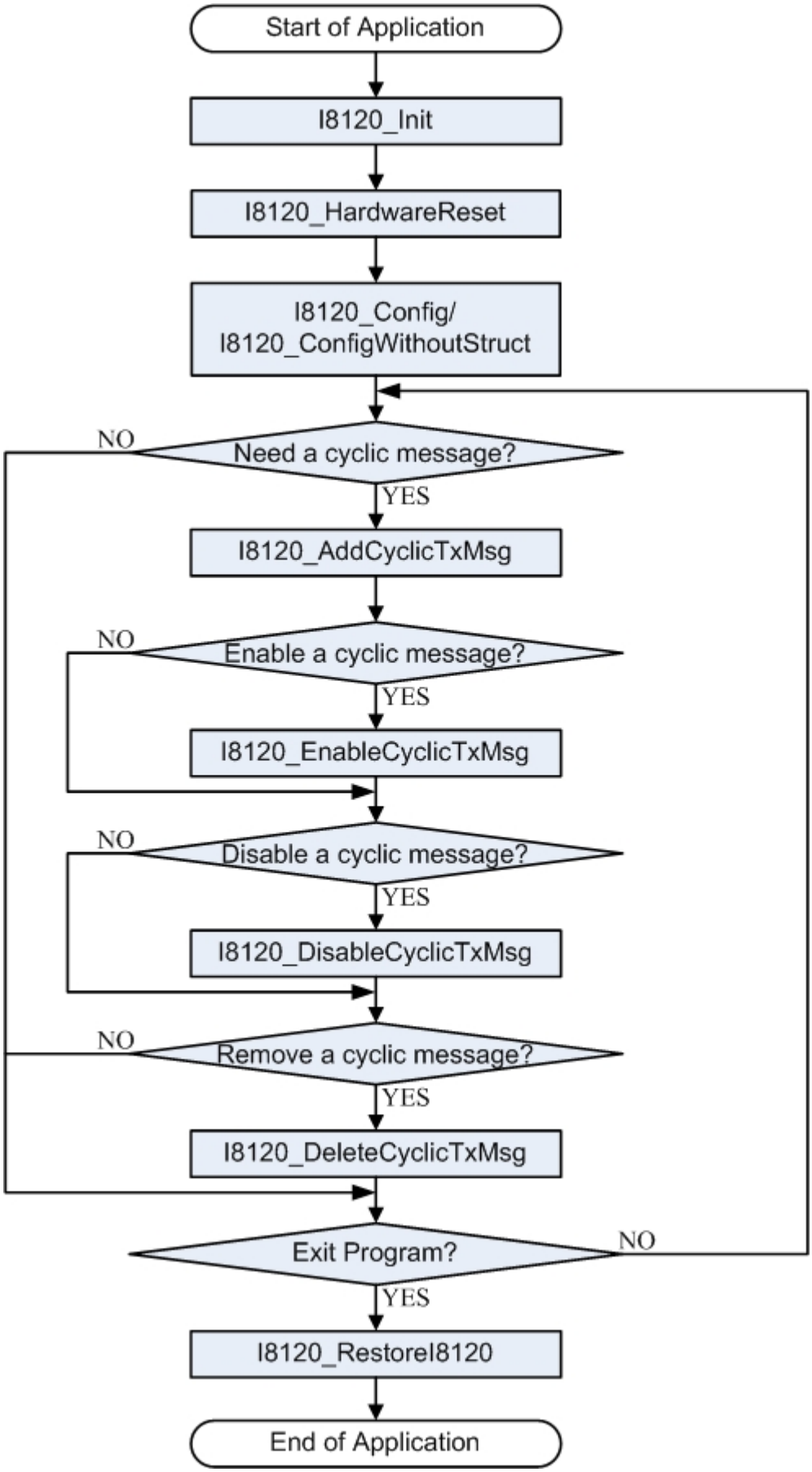


Figure 3.6 Flowchart of Cyclic Transmitting CAN Massages

3.3 Application Programming

When you want to design a WinPAC application, the VB.NET, C#.NET or eVC++ development environment may be needed. If users want to develop applications for LinPAC or iPAC, C/C++ language is the only one choice. Therefore, you need LinPAC SDK software for LinPAC applications, or need BC/TC for iPAC application. Users can free download the eVC++, WinPAC SDK, LinPAC SDK and TC++1.01 compiler through the following website.

eVC++:

http://www.microsoft.com/downloads/details.aspx?FamilyId=1DACDB3D-50D1-41B2-A107-FA75AE960856&displaylang=en

WinPAC SDK:

http://www.icpdas.com/products/PAC/winpac/download/winpac_8000/download_sdk.htm

LinPAC SDK and TC++ 1.01:

http://www.icpdas.com/download/download-list.htm

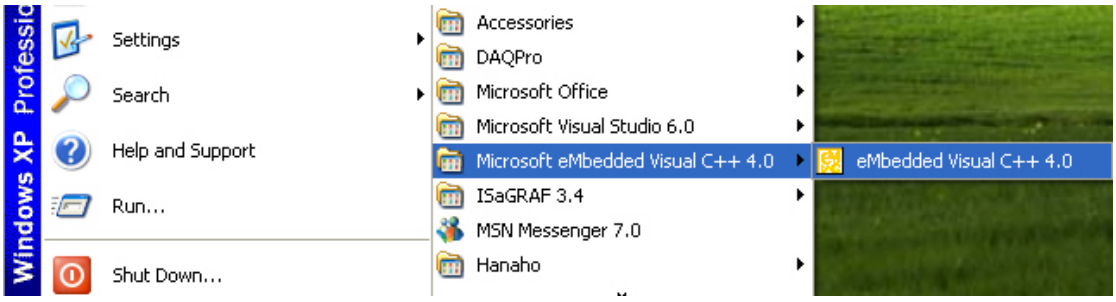
eVC++ Programming:

Step1: Download eVC++ 4.0 and install it in your PC.

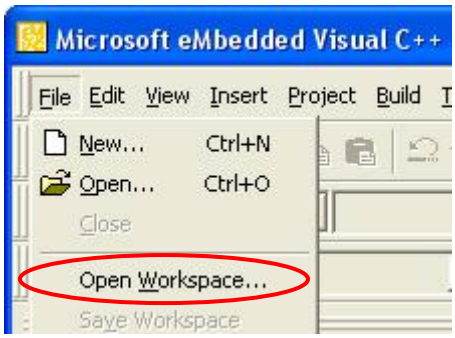
Step2: Download WinPAC SDK (for eVC) and install it in your PC.

Step3: If users want to know the details about WinPAC SDK, please refer to the SDK users' manual. Users can download it from our website: http://ftp.icpdas.com/pub/cd/winpac/napdos/wp-8x4x_ce50/document/sdk_document/

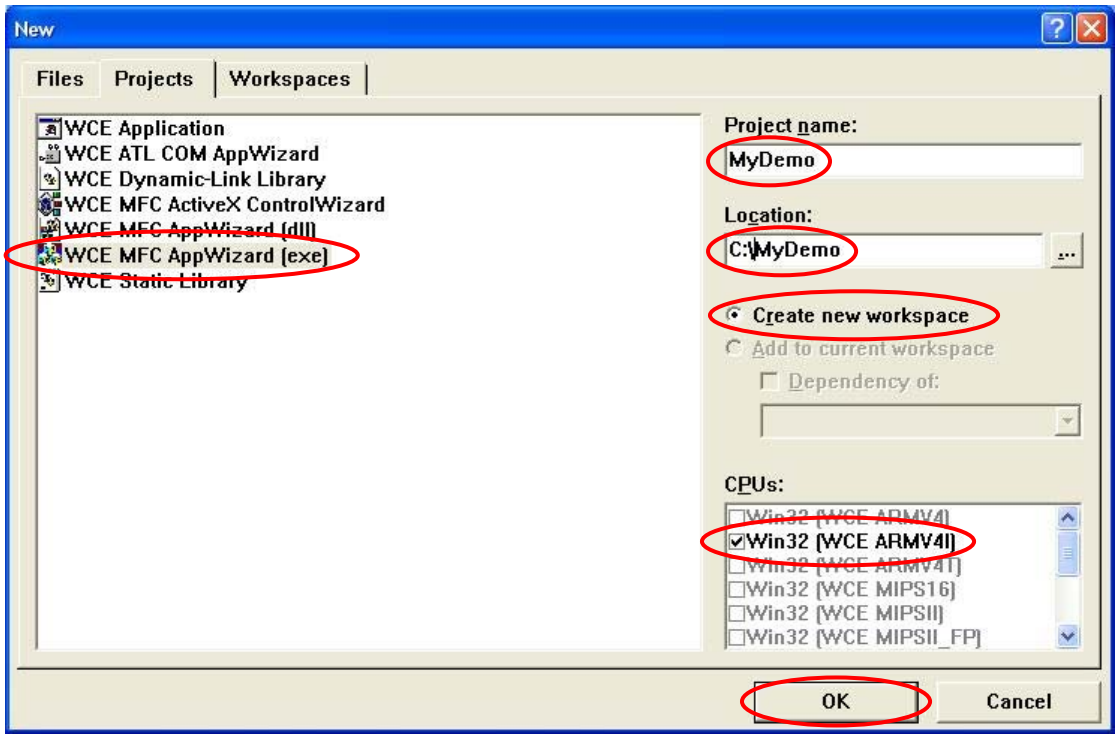
Step4: Execute the eVC++.



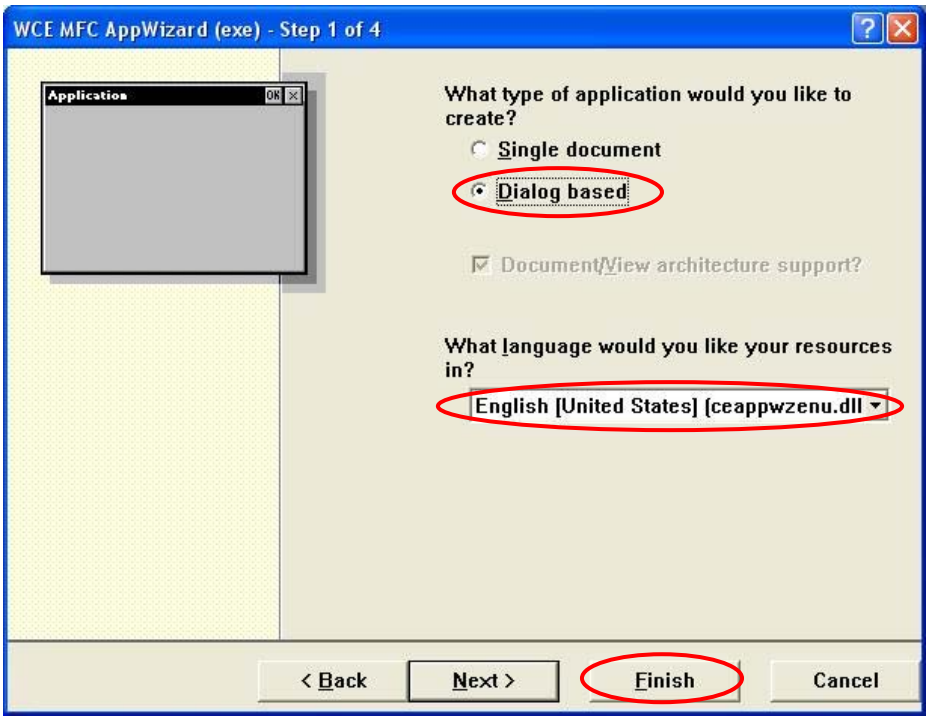
Step5: Click “File\New…” to create a new project.



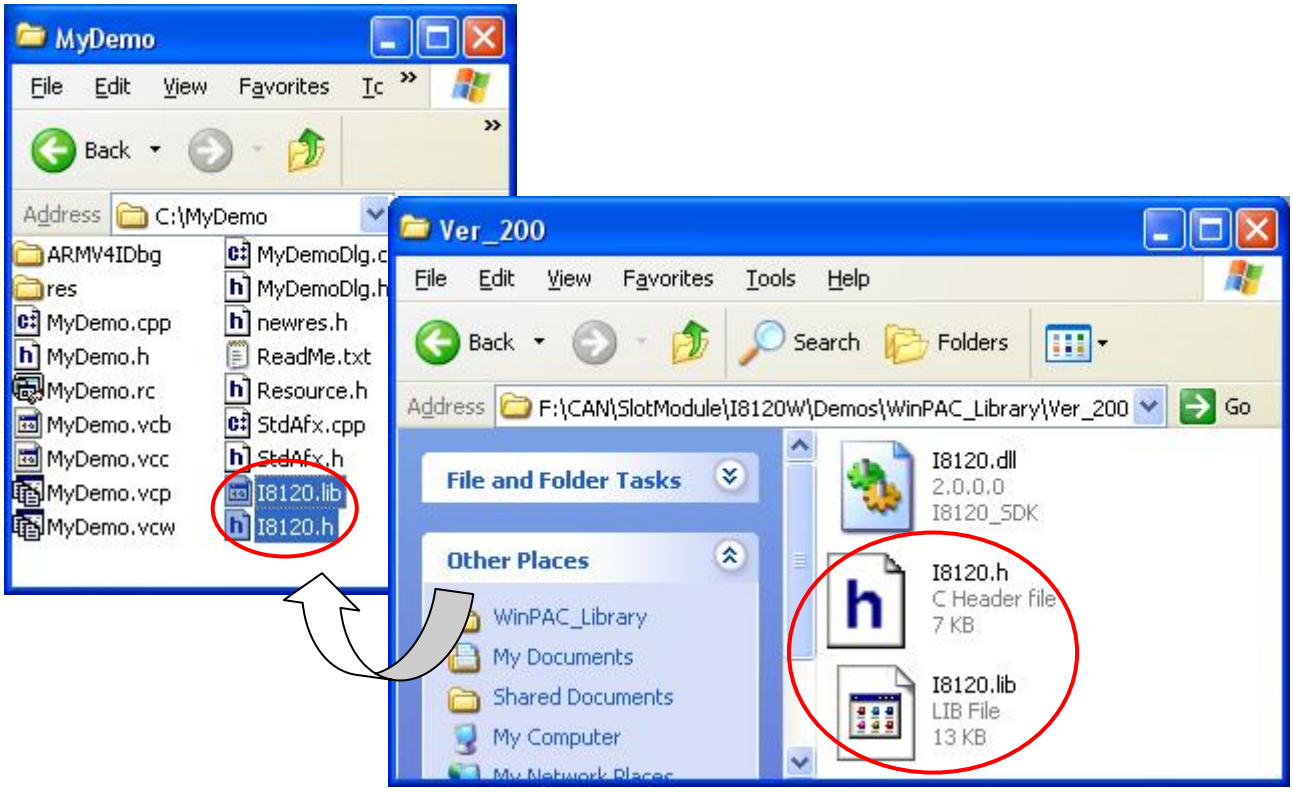
Step6: Select “WCE MFC AppWizard (exe)” to be the template of this project. The project name is “MyDemo”. The location of this project is “C:\MyDemo”. The CPU type in the CPUs field is set to “Win32 (WCE ARMV4I)”. Then, click OK to go on the next step.



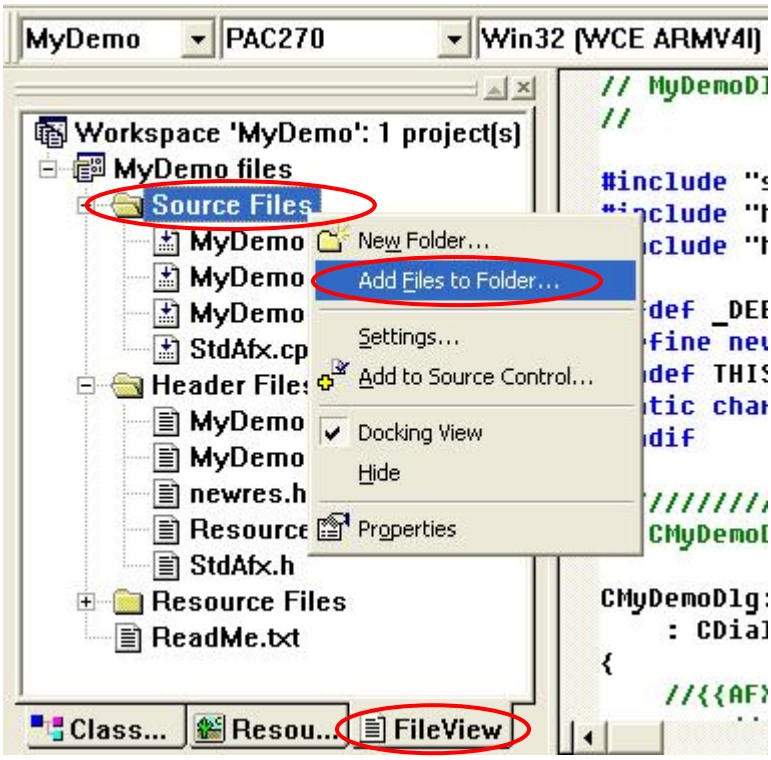
Step7: Select “Dialog based” item for this demo. Choose the language which you want to see in your resources file. Here, “English (United States)” item is used. Click “Finish” button to finish the project creation.



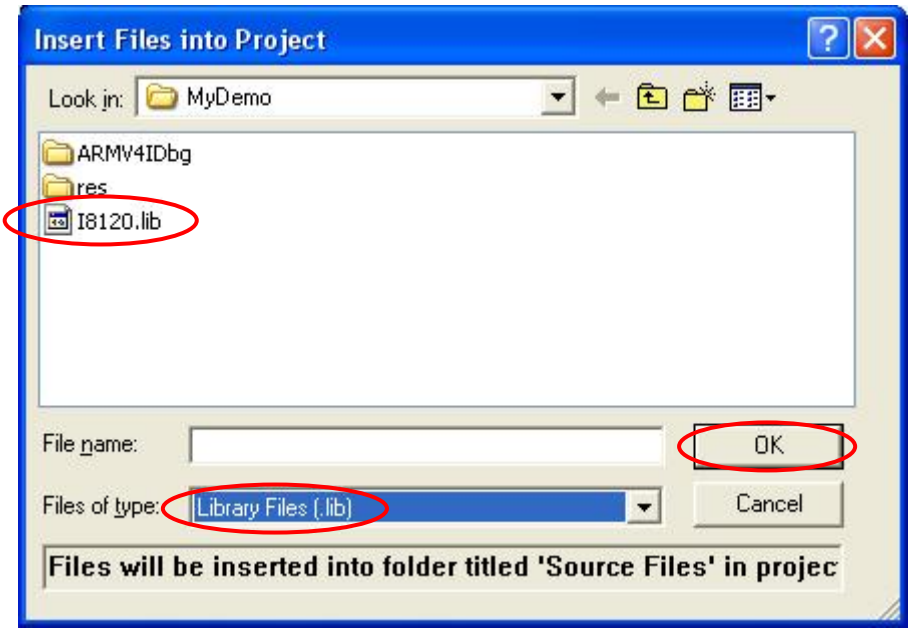
Step8: Copy the I-8120 library files “I8120.h” and “I8120.lib” into the MyDemo folder in disk C. You can find these files in the following path of CD: CAN\SlotModule\I8120W\Demos\WinPAC_Library\Ver_200.



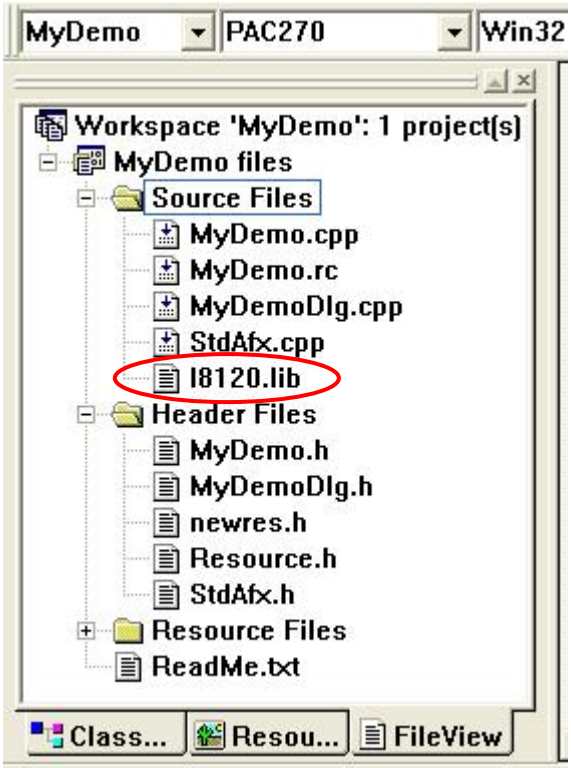
Step9: Select “File View” tag, and expand the tree view. Right click the “Source Files” folder icon and click “Add Files to Folder…” item.



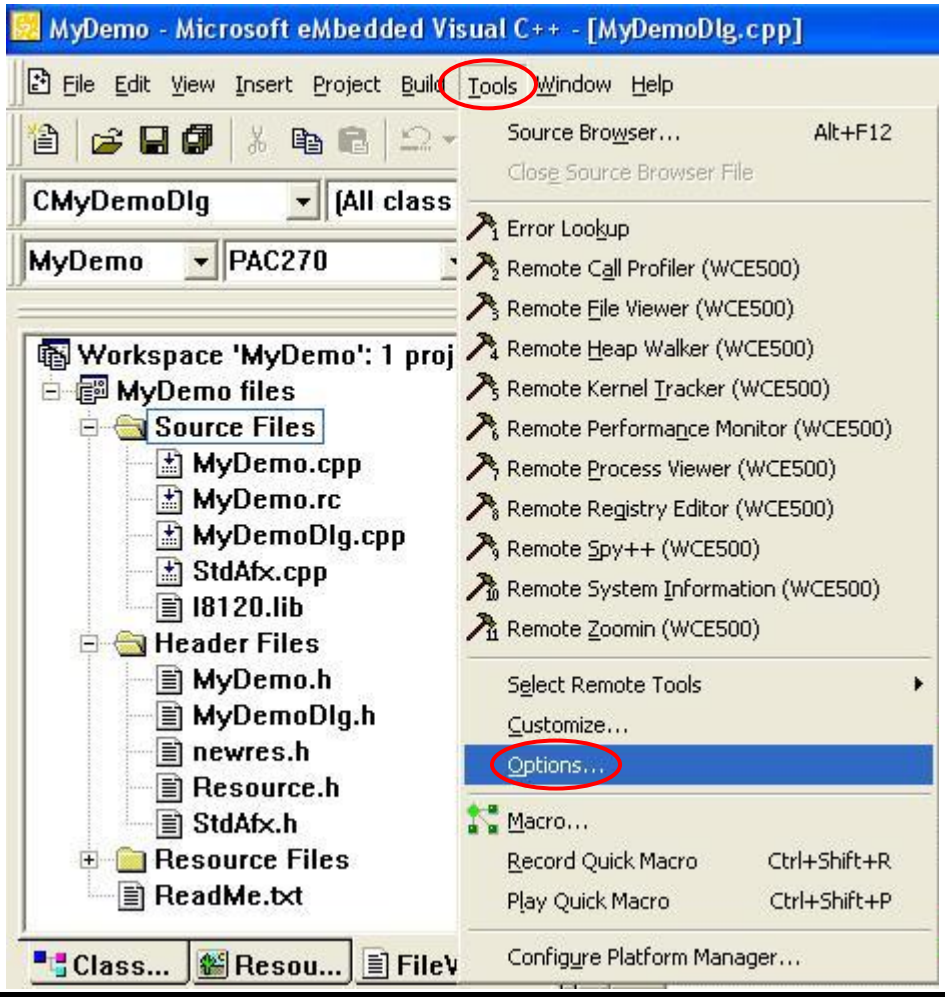
Step10: In the popup dialog, select “All Files (*.lib)” in the “Files of type” filed. Select the file I8120.lib and click OK button to add the library file of I8120W into MyDemo project.



Step11: After finishing the Step 10, the tree view of “File View” is shown below.



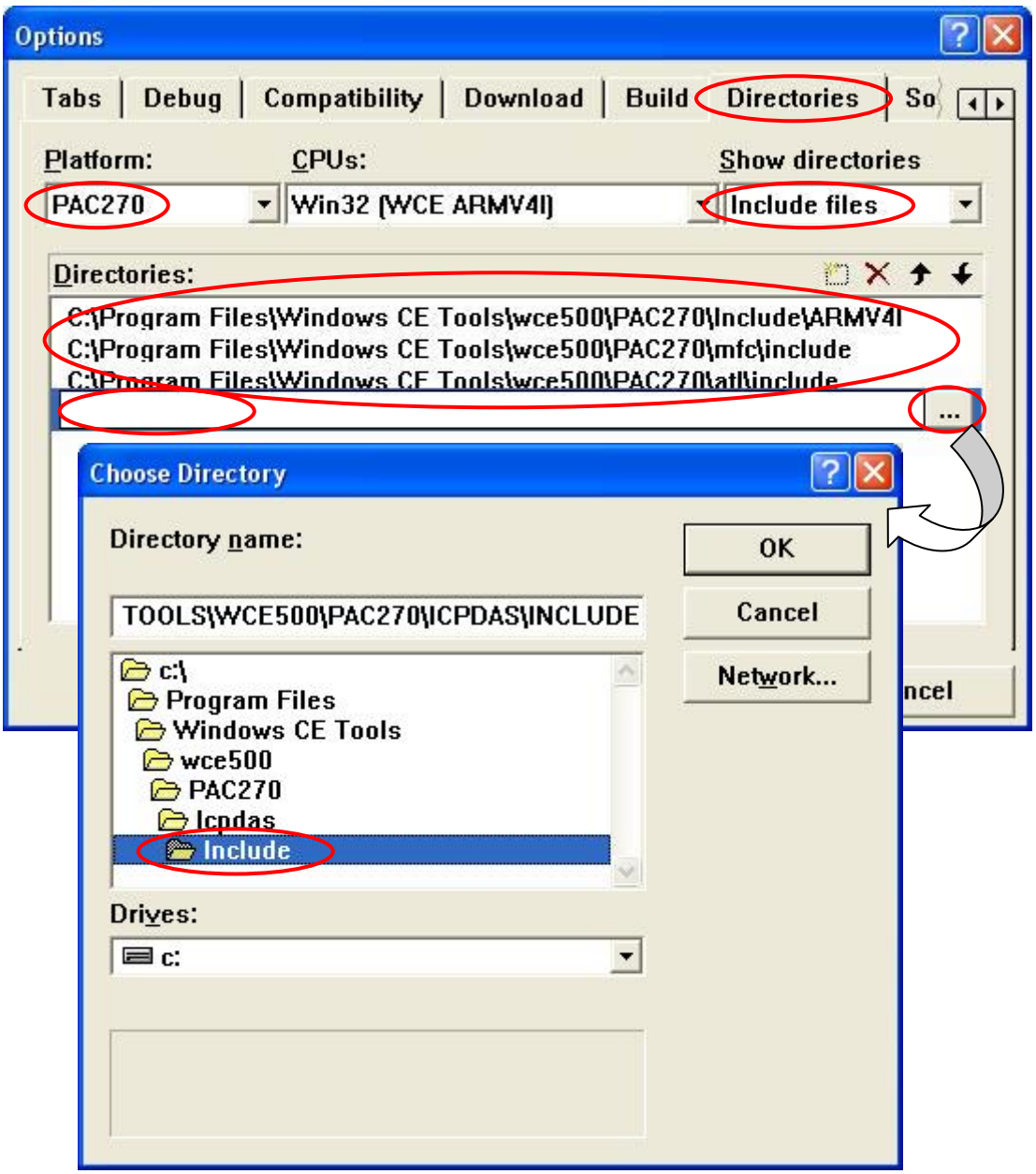
Step12: Click the “Tools\Options…” to set the “include” and “library” directory.



Step13: Set the “Platform” to “PAC270”. Set the “CPUs” to “Win32 (WCE ARMV4I)”. Select the “Include files” for the “Show directories” combo box. Set the “Directories” as follow:

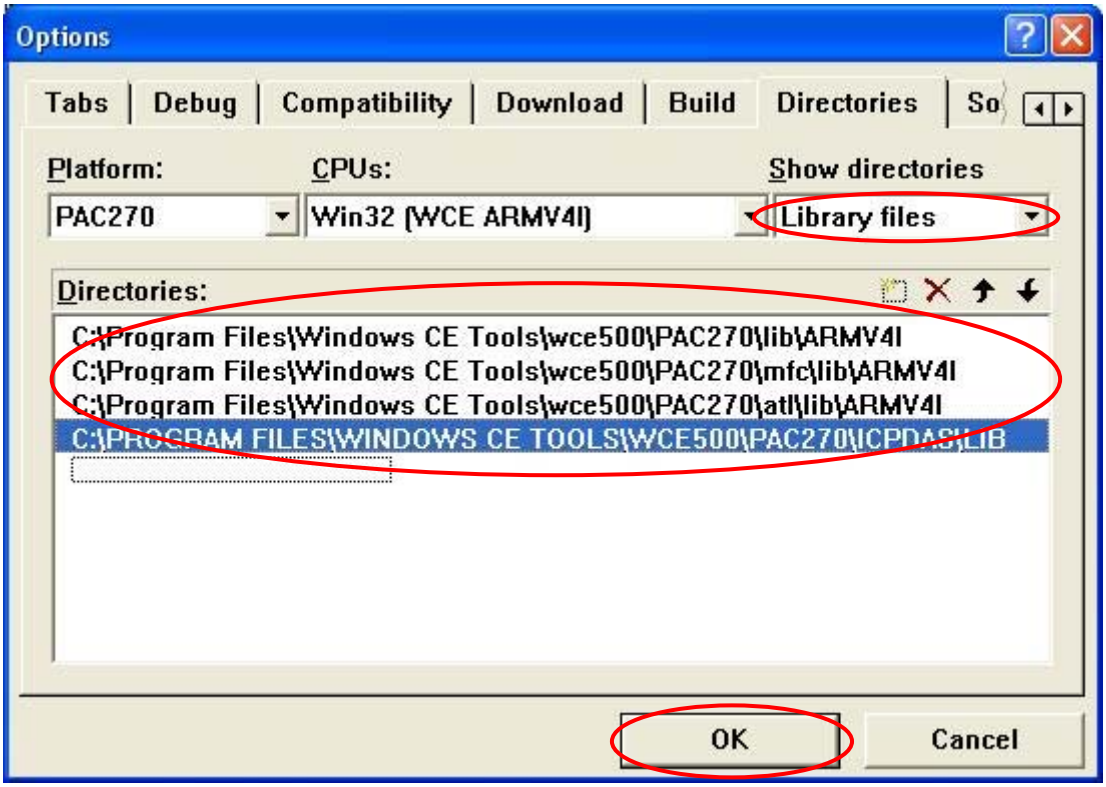
```
C:\Program Files\Windows CE Tools\wce500\PAC270\include\ARMV4I
C:\Program Files\Windows CE Tools\wce500\PAC270\mfc\include
C:\Program Files\Windows CE Tools\wce500\PAC270\ati\include
C:\Program Files\Windows CE Tools\wce500\PAC270\ICPDAS\include
```

If users want to add a new path in the “Directories”, double-click in the empty filed of the “Directories” and click “…” button to select the proper folder which you want to set. Then, click OK to continue.



Step14: Select the “Library files” for the “Show directories” combo box. Refer the Step 13 to set the “Directories” as follows:

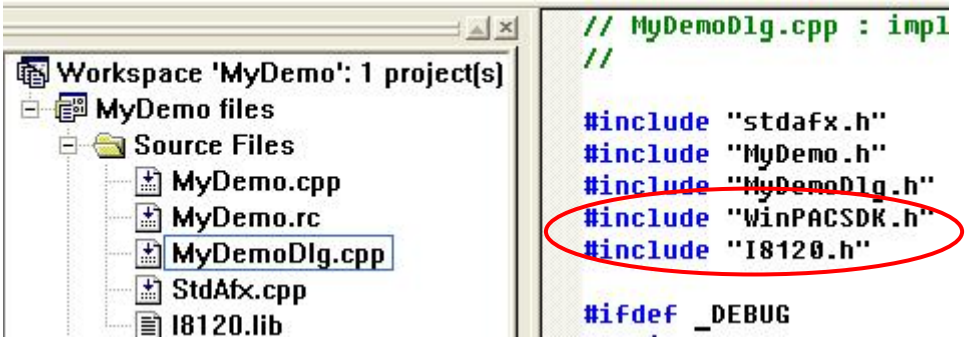
```
C:\Program Files\Windows CE Tools\wce500\PAC270\lib\ARMV4I
C:\Program Files\Windows CE Tools\wce500\PAC270\mfc\lib\ARMV4I
C:\Program Files\Windows CE Tools\wce500\PAC270\ati\lib\ARMV4I
C:\Program Files\Windows CE Tools\wce500\PAC270\ICPDAS\lib
```



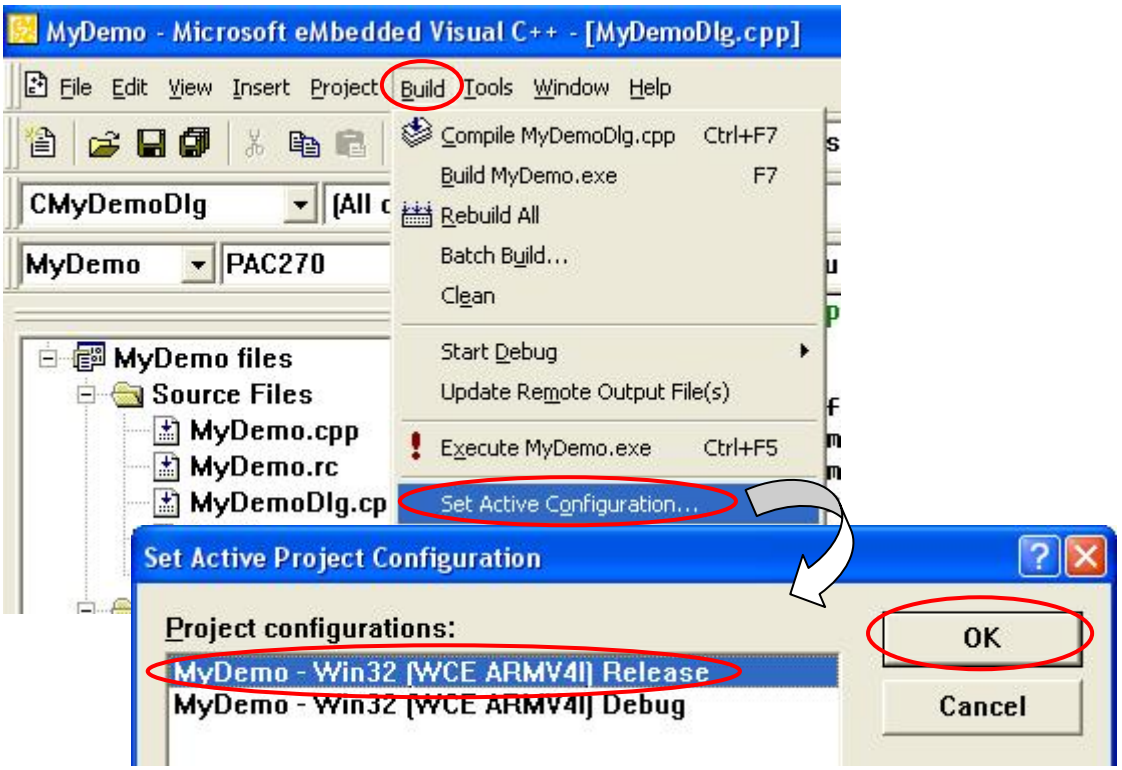
Step15: Program users’ application in the eVC++ IDE. When programming the MyDemoDlg.cpp, users must include the “WinPACSDK.h” and “I8120.h” to use APIs of WinPAC and I8120. The syntaxes are shown below.

#include “WinPACSDK.h”

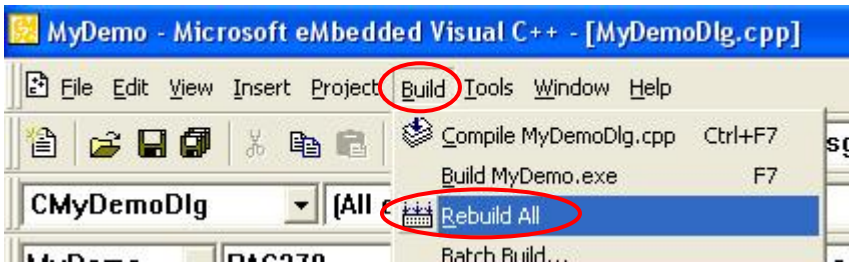
#include “I8120.h”



Step16: When finishing the program, Click “Build\Set Active Configuration” to select the project configuration to “Win32 (WCE ARMV4I) Release”.



Step17: Click “Build\Rebuild All” to build an execution file.



Step18: Use ftp method to copy the MyDemo.exe and I8120.dll to your WinPAC. The MyDemo.exe and I8120.dll must be put in the same folder. You can find the I8120.dll in the following path of CD: “CAN\SlotModule\I8120W\Demos\WinPAC_Library\Ver_200”. Then, run the MyDemo.exe on the WinPAC. If you want to know the details about how to use WinPAC, please refer to the following website:

http://www.icpdas.com/products/PAC/winpac/download/winpac_8000/download_documents.htm

3.4 Introduction of I8120W_Utility Tool

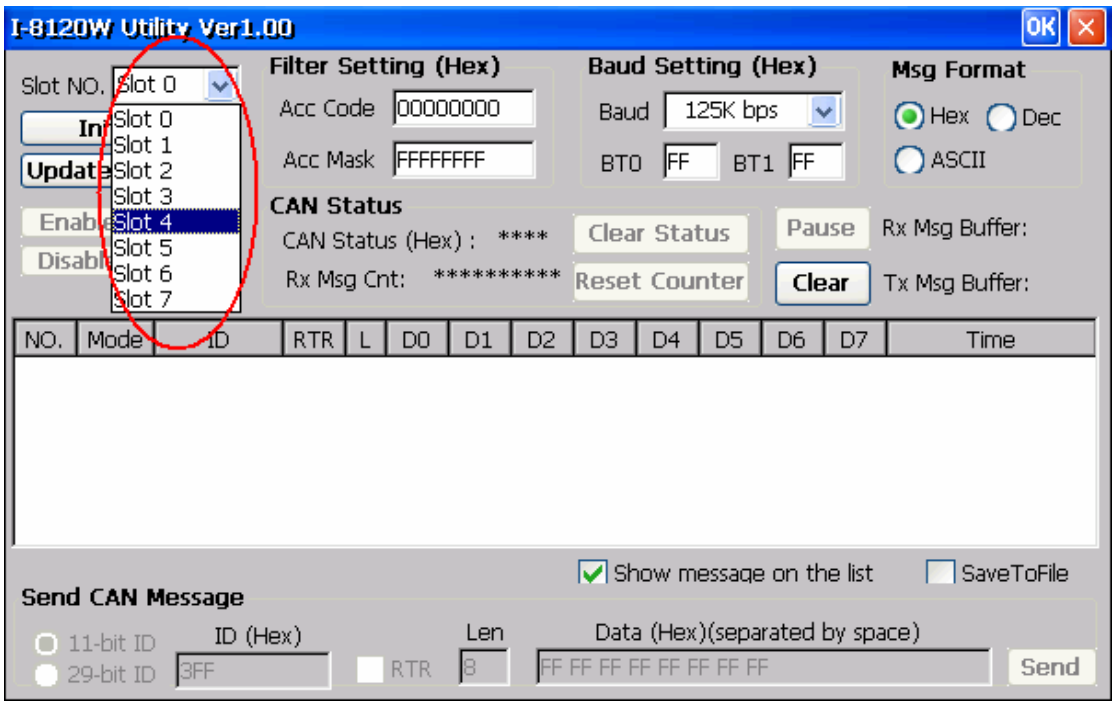
I8120W_Utility is designed for I-8120W on WinPAC/LinPAC/iPAC main control unit. It provides some useful functions when users want to update default firmware or download the user-defined firmware. The following section shows you how to use it on your main control unit.

I8120W_Utility for WinPAC:

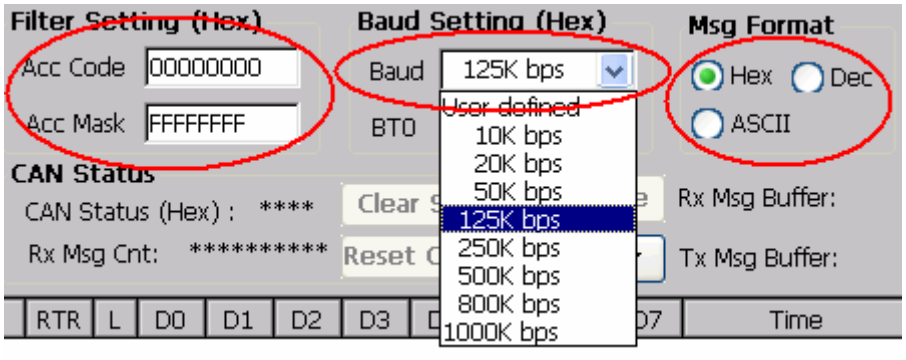
Beside the function of download firmware, I8120W_Utility on WinPAC also provides the functions to monitor/access the CAN messages. Users can find it in the path CAN\SlotModule\I8120W\Tools\WinPAC of the Field Bus CD. When you want to use it, you need to put the I8120.dll and the I8120W_Utility in the same folder. This file can be found in the path CAN\SlotModule\I8120W\Demos\WinPAC_Lib\Ver_200\.

CAN Network Access:

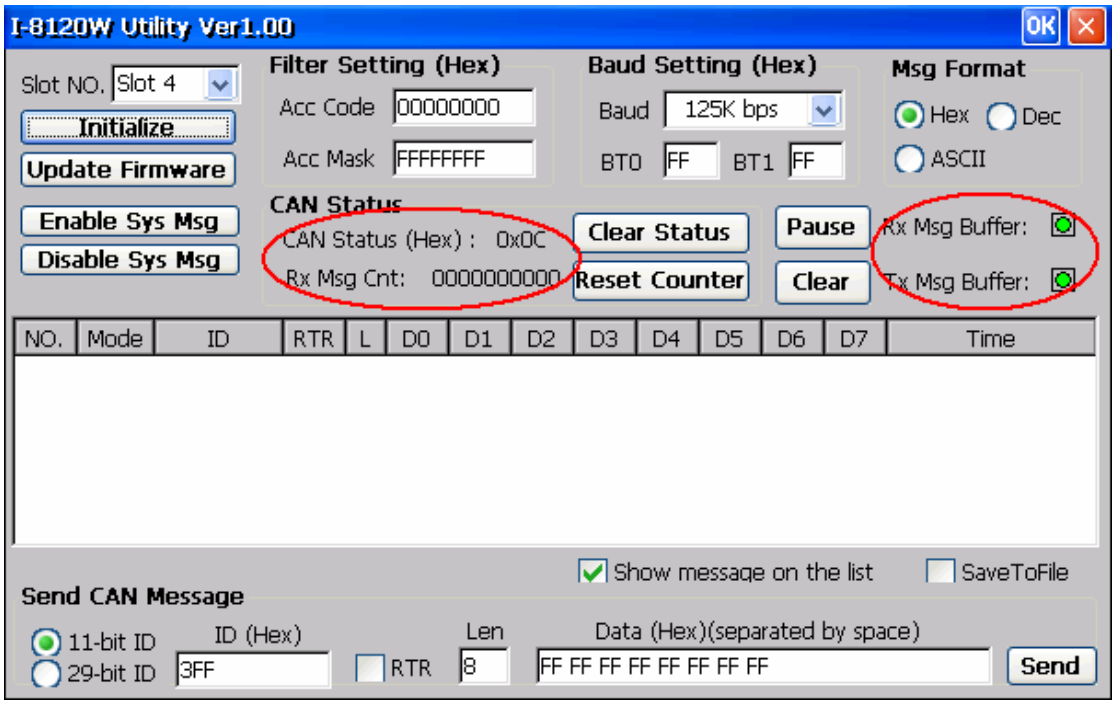
When the utility boots up, users can choose the proper No. of the slot which has plugged the I-8120W.



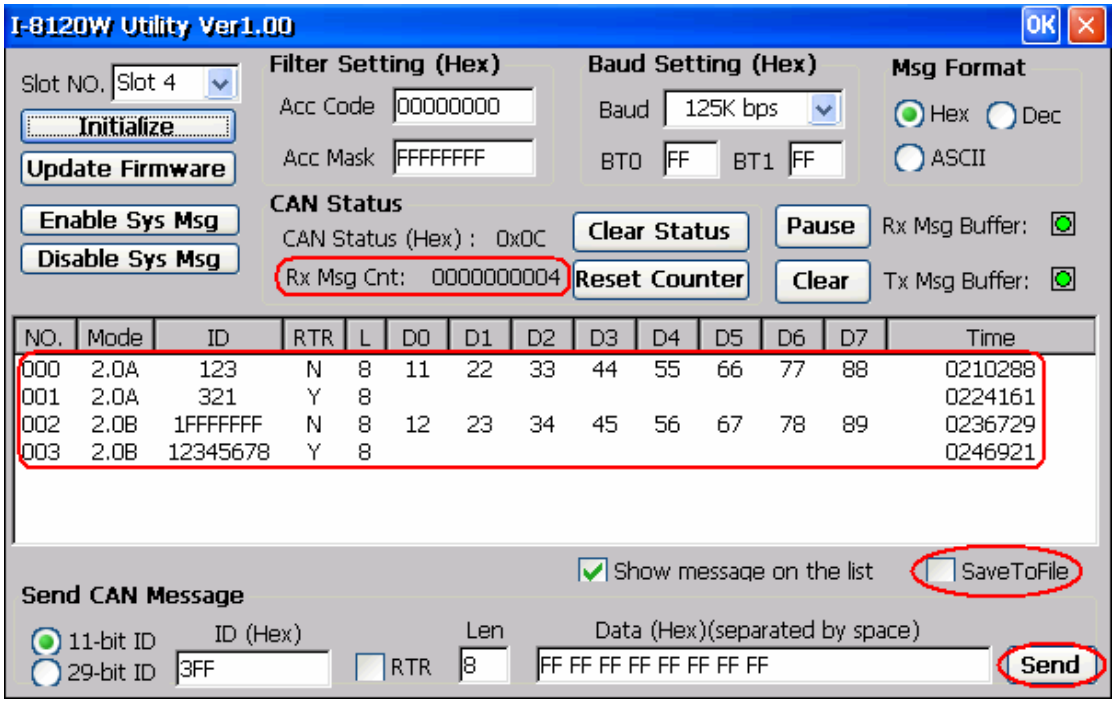
If users want to analysis the CAN network, please set the proper parameters of message filter, baud, and message format, then click the Initialize button. For the detail about how to use message filter and user-defined baud, please refer to the description of I8120_Config function. Afterwards, the utility will be shown as follows.



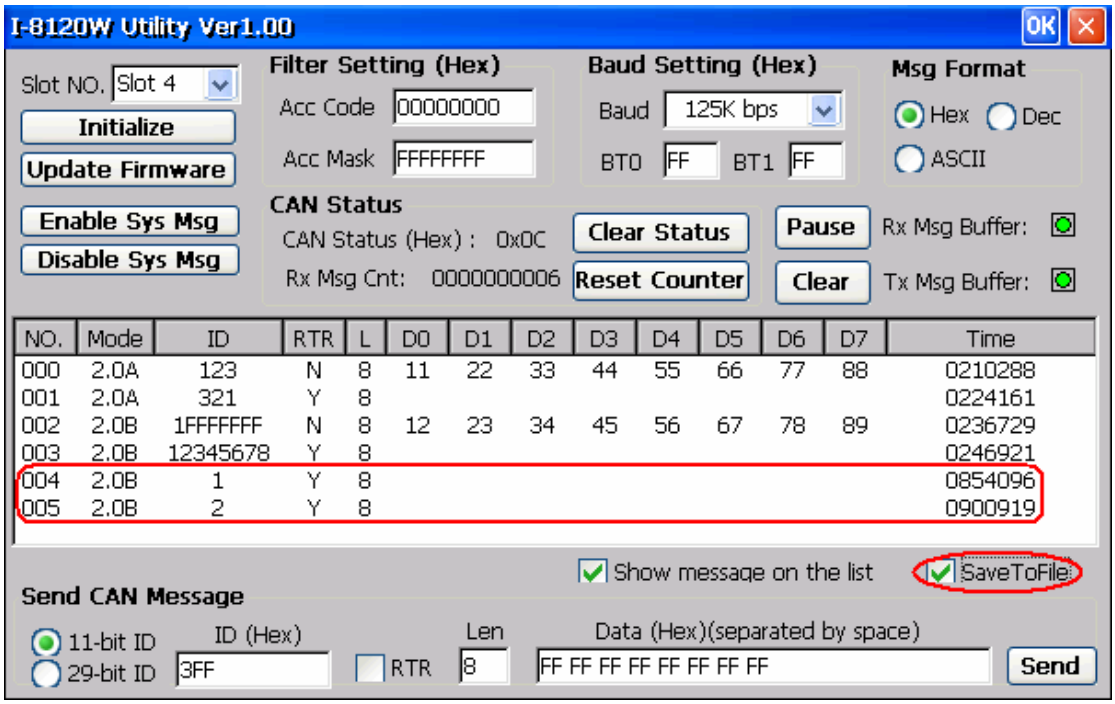
After finishing the configuration, users can see the CAN status is 0x0C if CAN controller works normally. The Rx and Tx message buffers status also shows green light. If you get the red lights, it means the message buffers are overflow. When I8120W_Utility receive any CAN message, the value of “Rx Msg Cnt” will be increased.



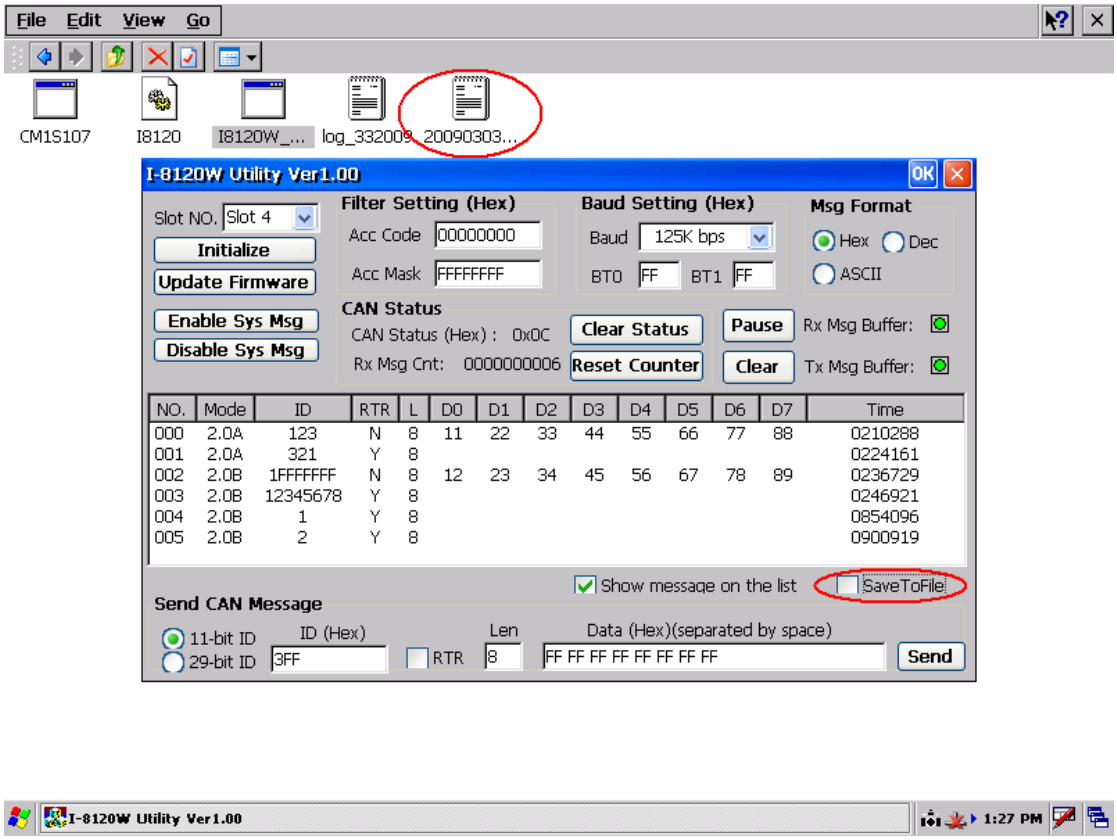
When the messages are received, they will be shown on the list. Users can also use “Send” button to send a CAN message. Furthermore, users can use “SaveToFile” checkbox to save the data into a .txt file. When users enable the checkbox, the following received messages will be saved.



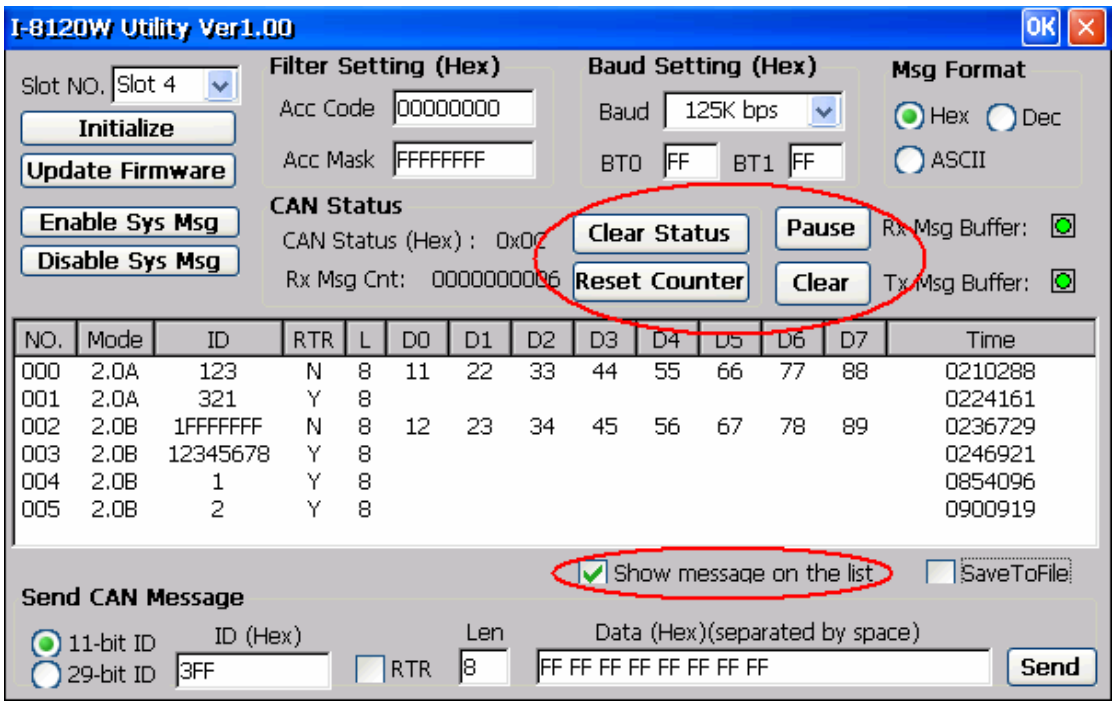
When the “SaveToFile” checkbox is enabled, assume that there are two messages received by I8120W_Utilty. These two messages will be saved into the .txt file.



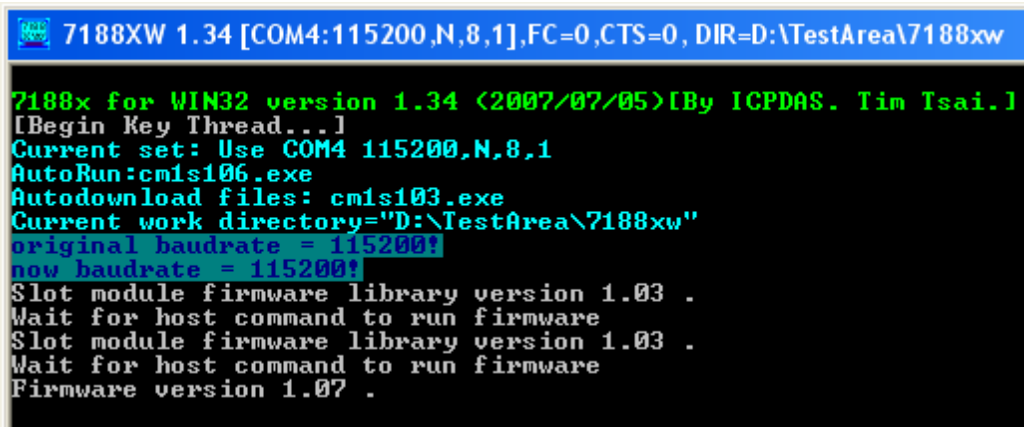
If users want to stop the data logger, disable the “SaveToFile” checkbox. At the same time, the logger file will be shown on the same path of the I8120W_Utility.



If the CAN network has some problem or the messages buffers are overflow, users can click “Clear Status” button to recover the CAN status, Rx Msg Buffer, and Tx Msg Buffer. The “Reset Counter” is used to clear the “Rx Msg Cnt”. If users click “Pause” button”, I8120W_Utiltiy will stop to receive the CAN messages. Click “Clear” button can clear the receiving list. All the positions of these buttons are shown as the screen shot on the next page. Because the display capability of WinPAC-8000, I8120W_Utility may receive the 20 frames per second without data lose. Users can disable the “Show messages on the list” checkbox and enable “SaveToFile”. The receiving speed can be up to 200 frames per second.

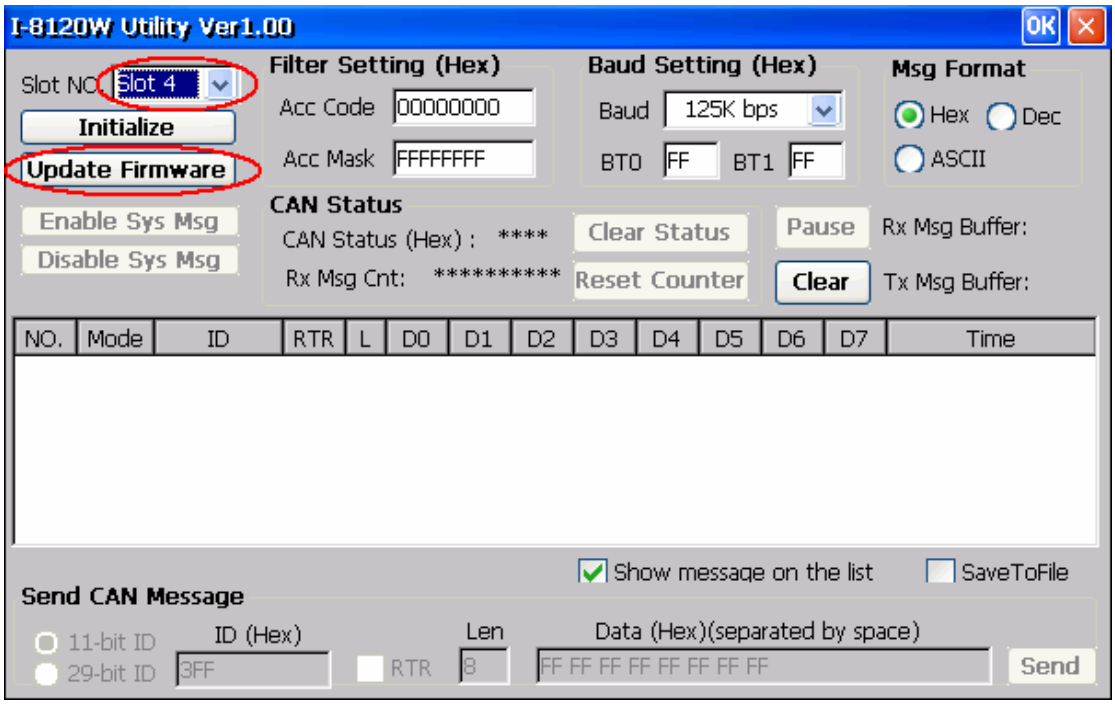


Sometimes, when users initialize the I-8120W, the system information may be shown to debug tool description in section 3.7. Users can use button “Enable Sys Msg”/“Disable Sys Msg” to enable/disable the system information.

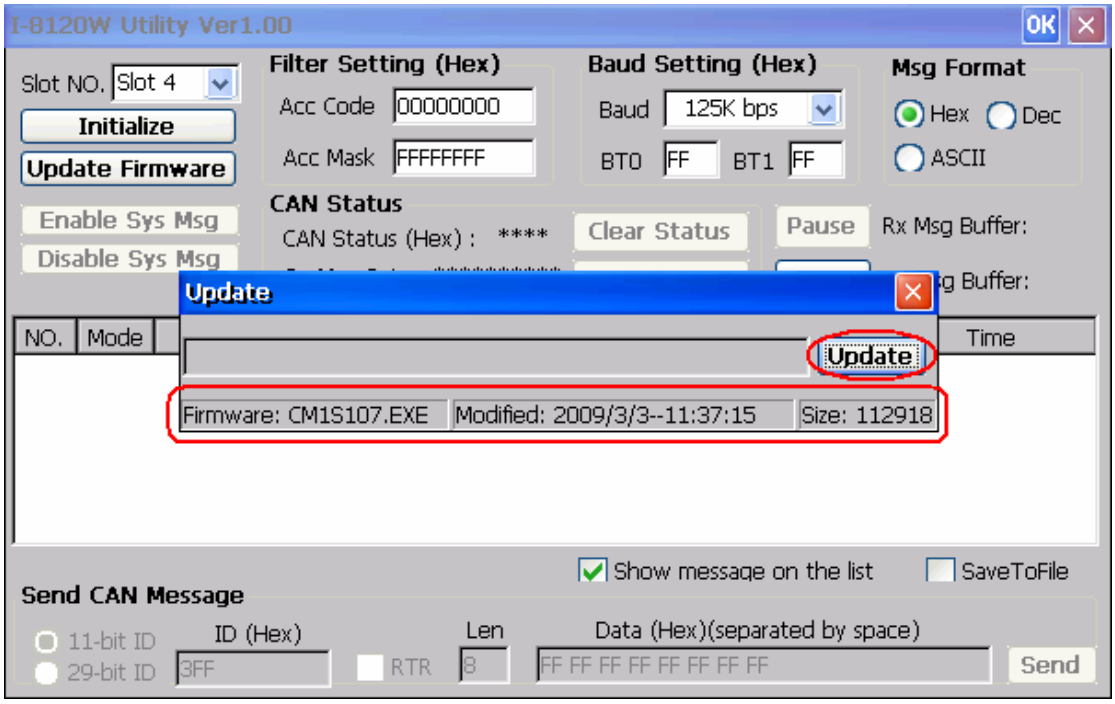


Program download:

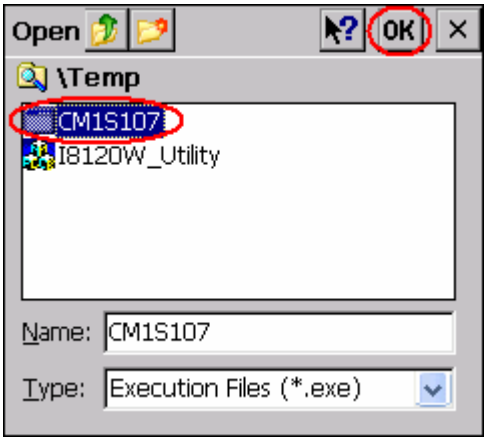
If users want to update the firmware of I-8120W or download the user-defined firmware into I-8120W, the following steps may be a good reference. First, choose the proper No. of the slot which has plugged the I-8120W. Then click the “Update Firmware” button.



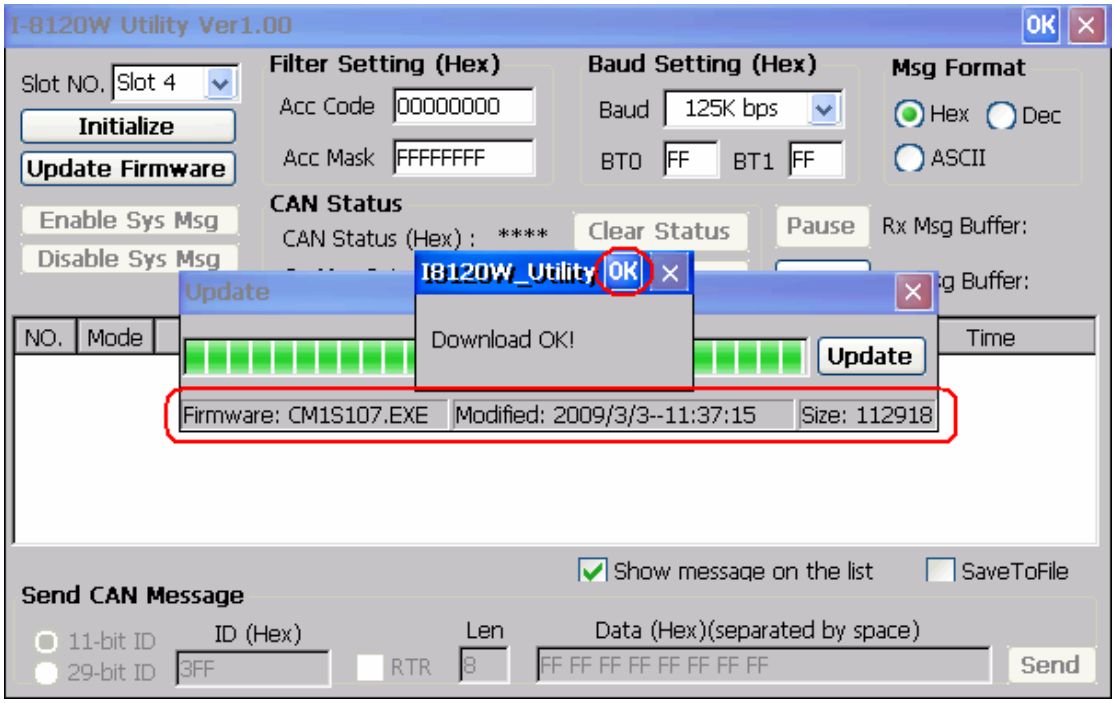
The download dialog will be pop up. Users can see the firmware name, modified date, and file size of the firmware stored in the I-8120W. Then click “Update” button to continue.



In the browsers, select the file which you want to download. Then click “OK” button to go on the download procedure. Take a note that when users click “OK” button, the download procedure will be started. The original firmware stored in the I-8120W will be killed.



When the procedure is finished, users can see the firmware information of new firmware. Click “OK” button to close the download dialog. Afterwards, the new firmware will be run automatically.



3.5 Basic concept of User-defined Firmware Programming

If users just apply default firmware for their application, this section can be ignored. This section describes about how to build a user-defined firmware. A CAN application can be implemented corresponding to the good cooperation of WinPAC/LinPAC/iPAC application and the user-defined firmware. Generally speaking, the user-defined firmware processes the part of CAN communication protocol and some algorithms of input and output. The WinPAC/LinPAC/iPAC application gets the processed data from user-defined firmware and shows them on the HMI interface, or gives a command to user-defined firmware to do some specified process. The relationship between WinPAC/LinPAC/iPAC applications and the user-defined firmware is shown as the figure 3.7 on the next page.

The Figure 3.8 and 3.9 shows the basic flowchart of developing the user-defined firmware and corresponding WinPAC/LinPAC/iPAC applications. To develop the user-defined firmware, users can create a C/C++ project, and include several .c file and 186COMM.lib. Put the 4 callback functions in one of these .c file. Program the codes into these 4 callback functions. If necessary, build your functions and global variables. Then, compile this project, and you can get your user-defined firmware. Download it by using utility tool and test it. Afterwards, according to the functions of user-defined firmware, design your WinPAC/LinPAC/iPAC applications. We provide some communication functions in the firmware library, 186COMM.lib. By using these functions, users can communicate WinPAC/LinPAC/iPAC applications with user-defined firmware via DPRAM. Besides, firmware library also supports most functions of hardware on I-8120W, such as DPRAM accessing, EEPROM accessing, RTC access, timer function… and so forth. In the WinPAC/LinPAC/iPAC applications, the communication functions are also given by I8120.dll. Moreover, it also provides some useful functions, such as cyclic transmission engine, hardware reset function, SJA1000 configuration functions, DPRAM accessing… and etc.

If users want to develop the user-defined firmware, C/C++ language is the only one choice. Therefore, you need BC/TC for user-defined firmware. Users can free download the TC++1.01 compiler through the following website.

http://www.icpdas.com/download/download-list.htm

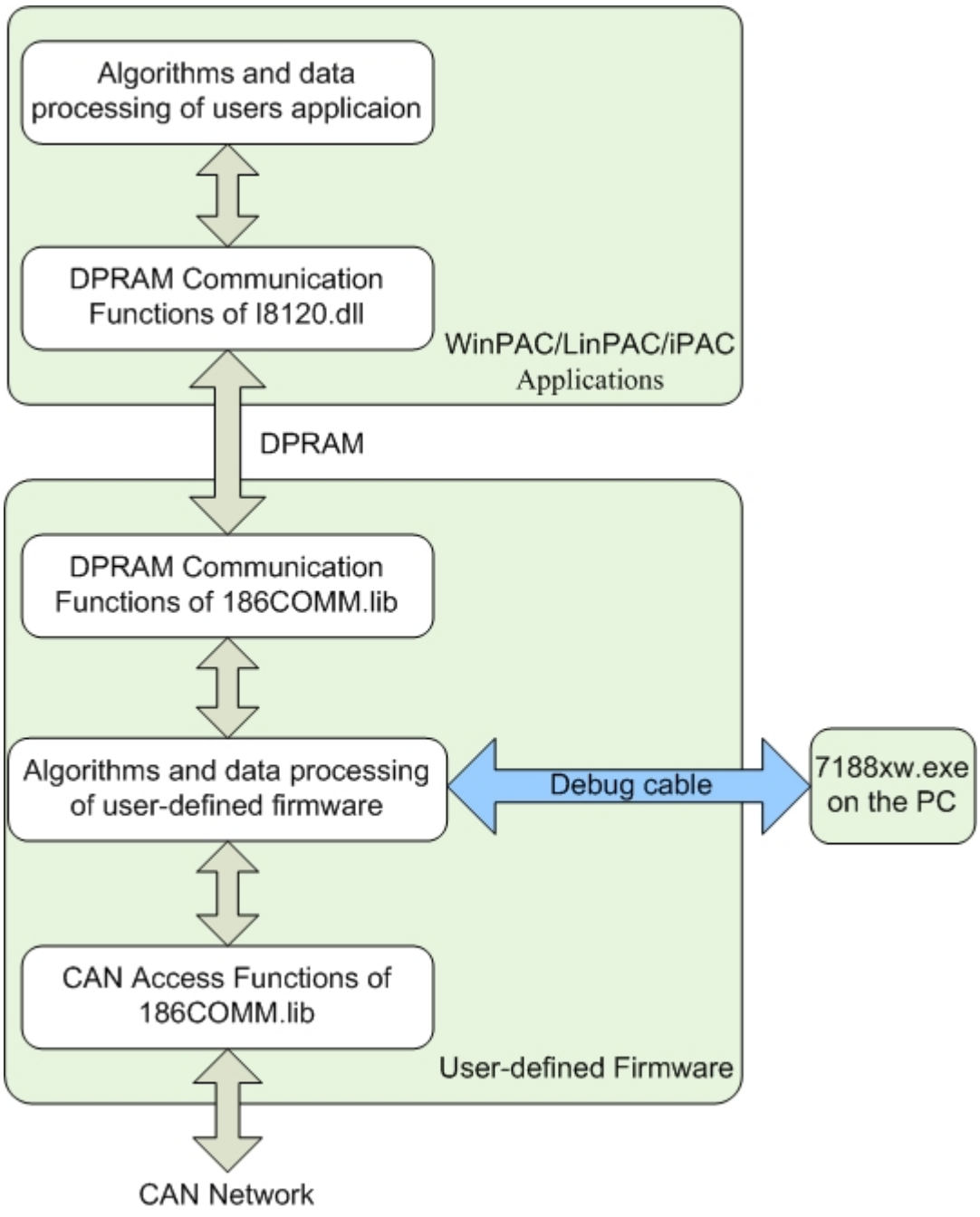


Figure 3.7 Relationship Between Applications & User-defined Firmware

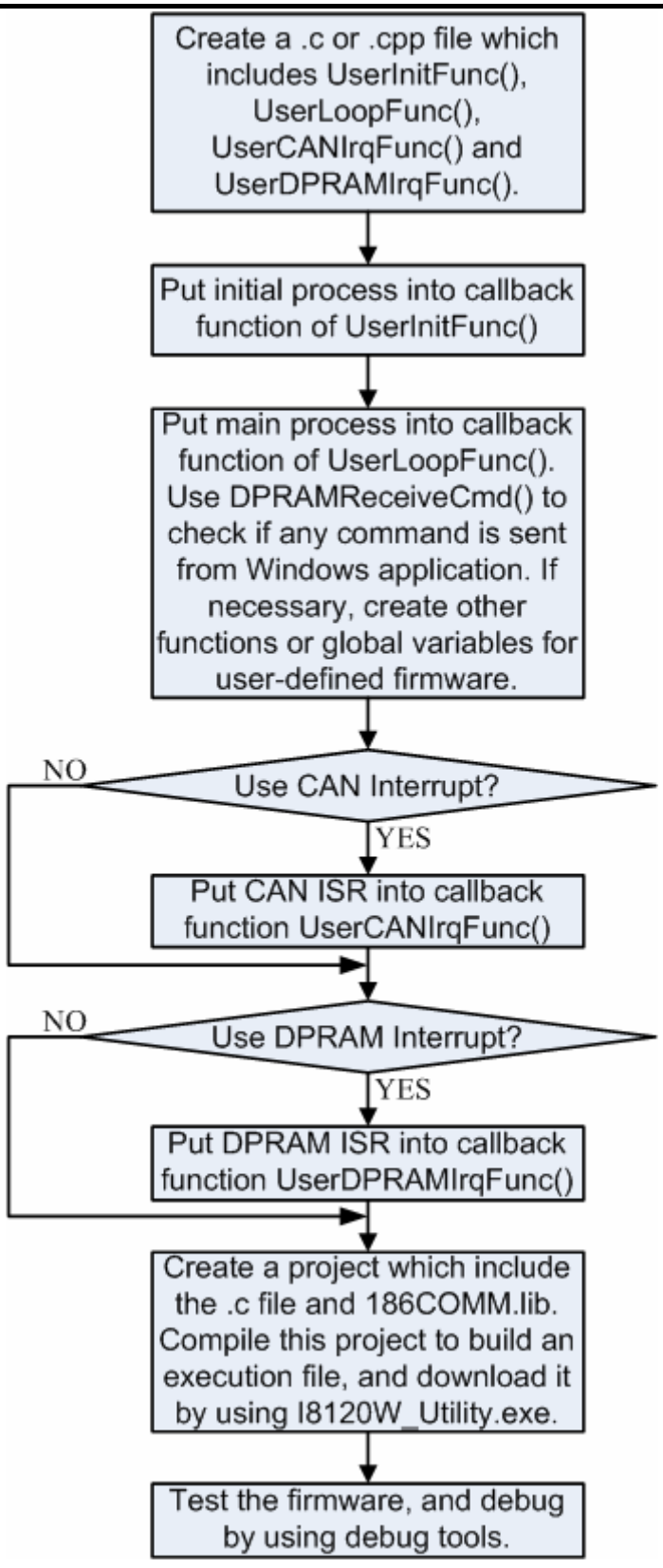


Figure 3.8 Development Procedure of User-defied Firmware

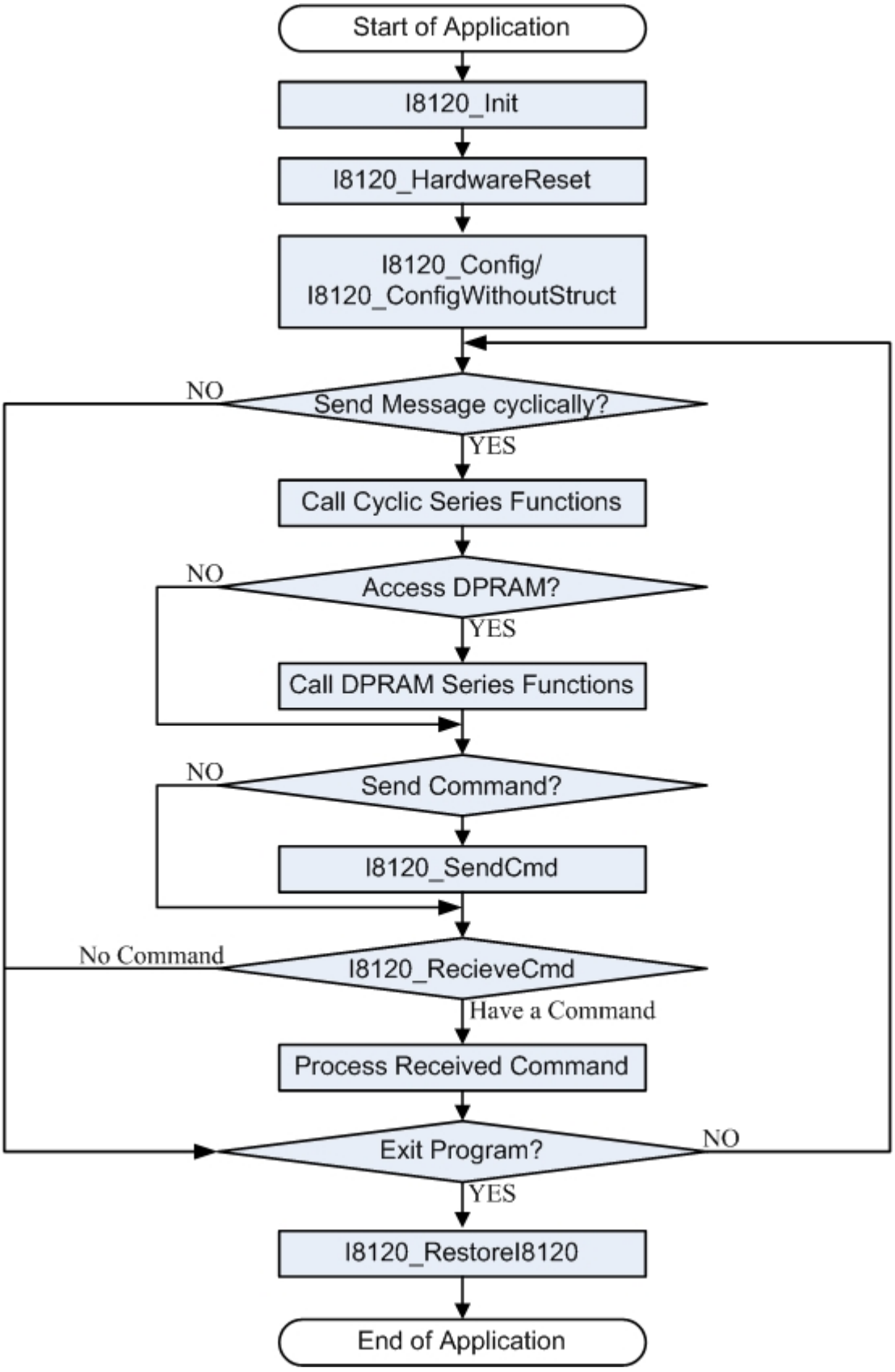


Figure 3.9 Procedure of Applications for User-defied Firmware

Briefs of the demo programs:

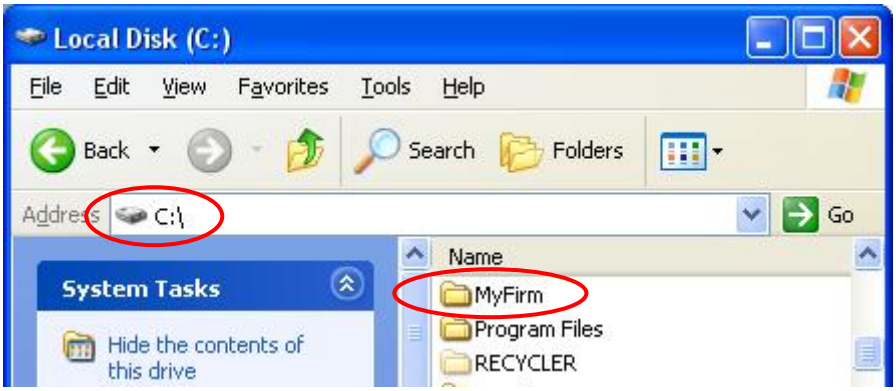
The following list shows all of the demos for user-defined firmware. These demos show two things: The one is how to program the firmware of I-8120W by using firmware library. Another is how to build a corresponding application on WinPAC/LinPAC/iPAC by using the I-8120W APIs. When users use the demos of user-defined firmware, the firmware of I-8120W in these demo folders are needed to download into the I-8120W firstly. Each demo folder has its own firmware of I-8120W. This firmware is written according to the application program in the same demo folder. After finishing the firmware download, users can use the demo of WinPAC/LinPAC/iPAC to communicate the firmware just download before.

```
|--\Demos                          → I-8120W demo programs
   |--\For_User_Defined_Firmware   → Folder for user-defined firmware
      |--\Firm_Lib                 → Firmware library of I-8120W
      |--\RxMsg                    → Demo for getting CAN messages
         |--\I8120W                → User-defined firmware for I-8120W
         |--\WinPAC                → Corresponding application for WinPAC
      |--\TxMsg                    → Demo for sending CAN messages
         |--\I8120W                → User-defined firmware for I-8120W
         |--\WinPAC                → Corresponding application for WinPAC
      |--\CANopen                  → Demo for a basic CANopen application
         |--\I8120W                → User-defined firmware for I-8120W
         |--\WinPAC                → Corresponding application for WinPAC
      |--\DevNet                   → Demo for a basic Device application
         |--\I8120W                → User-defined firmware for I-8120W
         |--\WinPAC                → Corresponding application for WinPAC
      |--\HostWDT                  → Demo of watchdog between Host and I-8120W
         |--\I8120W                → User-defined firmware for I-8120W
         |--\WinPAC                → Corresponding application for WinPAC
```

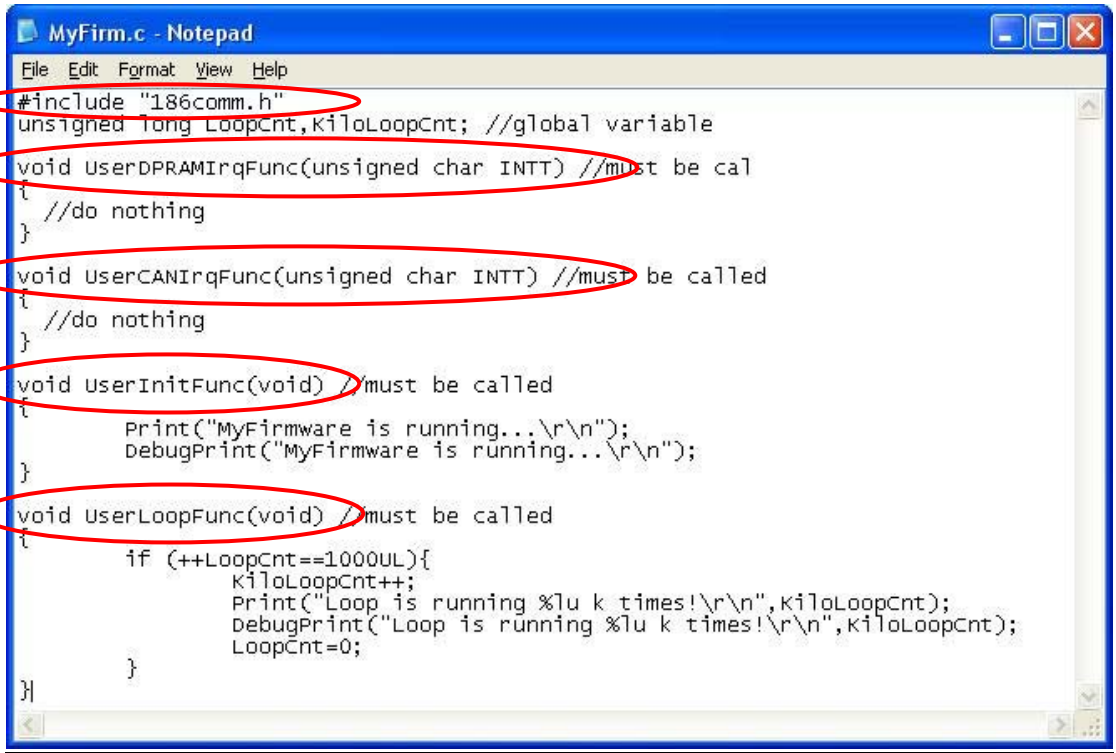
3.6 User-defined Firmware Programming

Here, it is considered that how to build an execution file with 186COMM.lib by using TC++1.01 compiler. It may be a good model for development a user-defined firmware. Before starting the step-by-step procedure, users need to install TC++1.01. Users can free download the TC++1.01 through the website: http://www.icpdas.com/download/download-list.htm

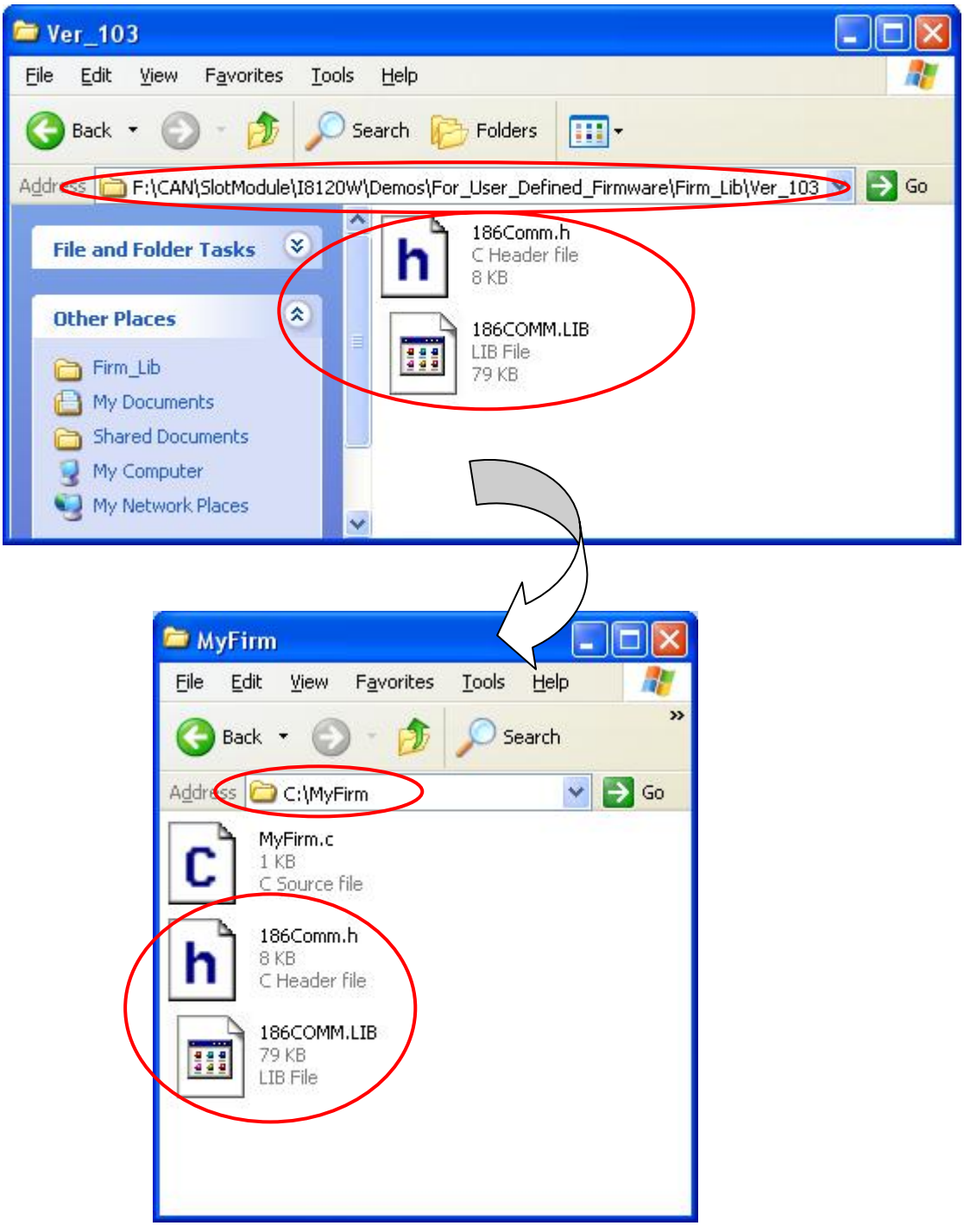
Step1: Create a folder named “MyFirm” in the C disk.



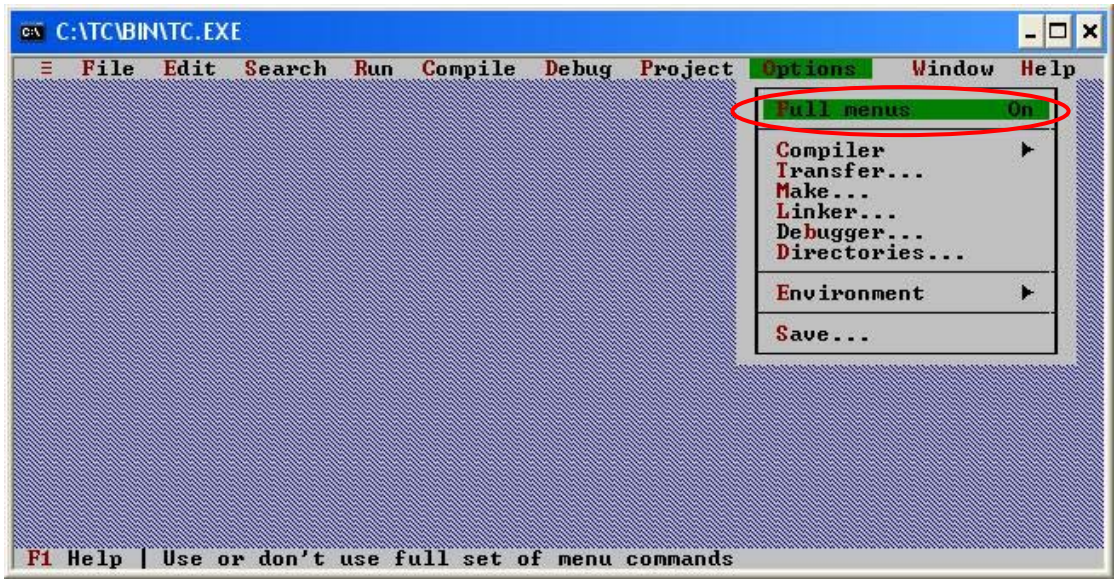
Step2: In the folder MyFirm, create a .c file and name it as “MyFirm.c”. Design the MyFirm.c file as follows. The 4 callback functions must be used in user-defined function.



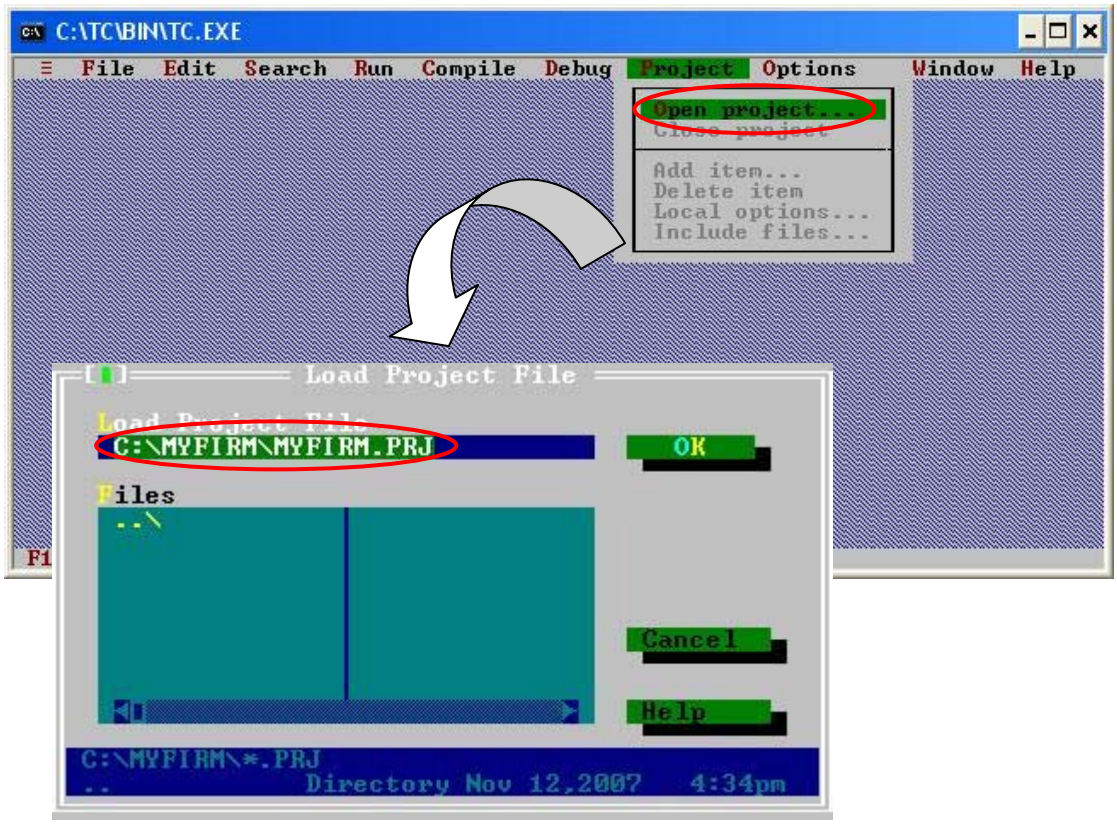
Step3: Copy 186COMM.lib file and 186COMM.h file into MyFirm folder. Users can find them with version 1.03 in the path CAN\SlotModule\I8120W\Demos\For_User_Defined_Firmware\Firm_Lib\ver_103 in “Field Bus” CD.



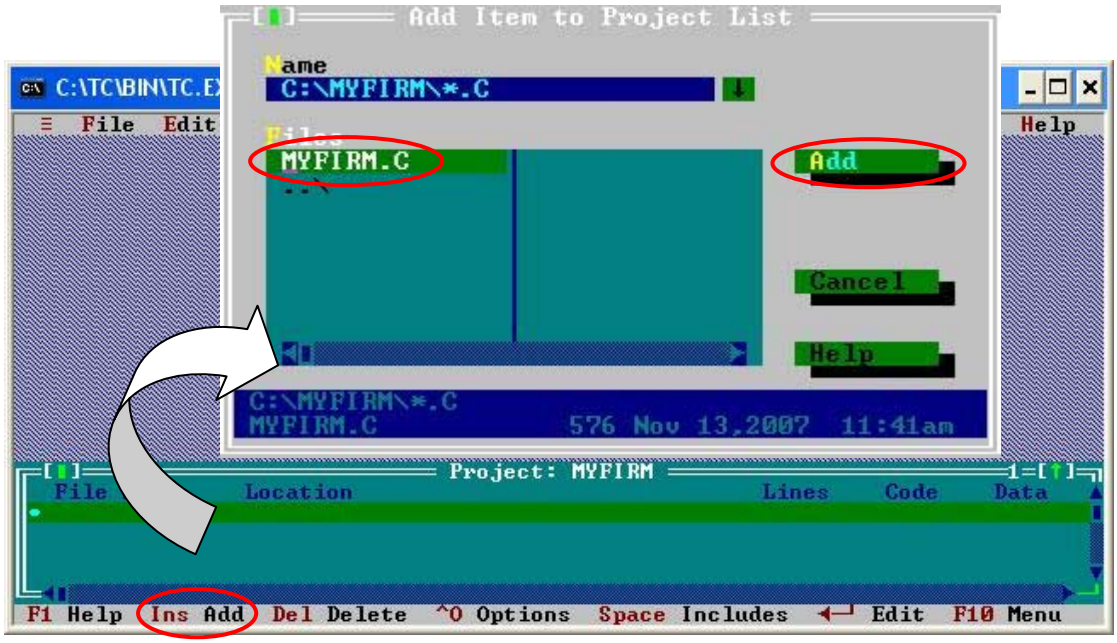
Step4: Run the TC++1.01 development environment. Click the “Options\Full menus” to expand the all functions list in the menus.



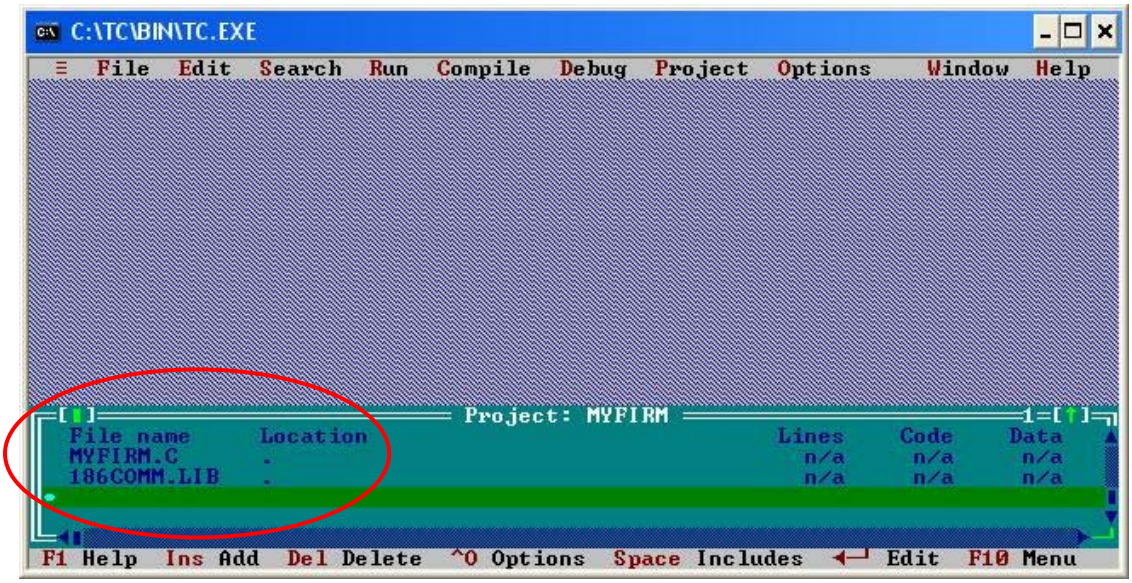
Step5: Click the “Project\Open project…” to create a new project. Input the project name “MyFirm.PRJ”, and click OK button to continue.



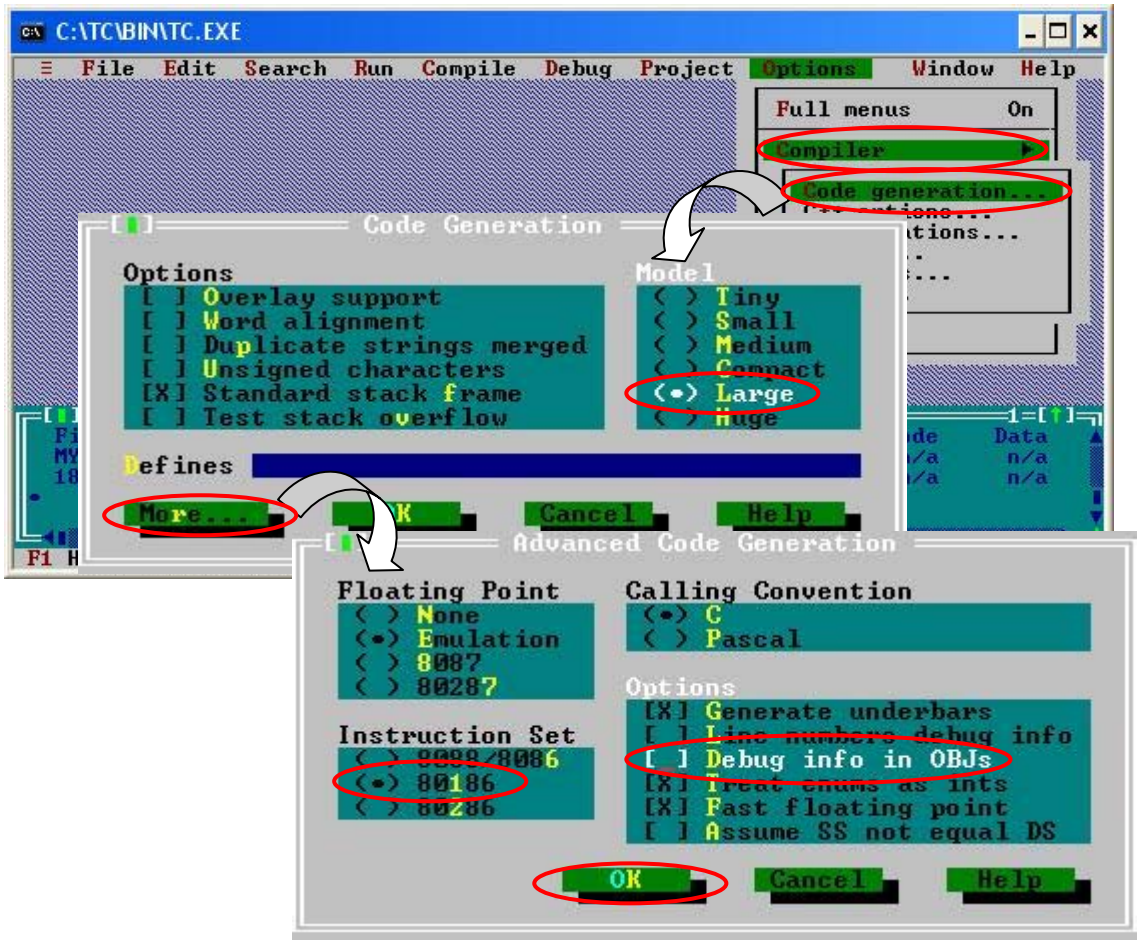
Step6: Click “Add” function on the bottom of TC++1.01 screen. Search all .c file by setting c:\MyFirm*.c in the Name field of popup window. Then, use the “Add” button to add the MyFirm’ .c file in to MyFirm project. Then, change the search command from “c:\MyFirm*.c” to “c:\MyFirm*.lib” in the Name field. Add the library files 186COMM.lib into MyFirm project by the same way.



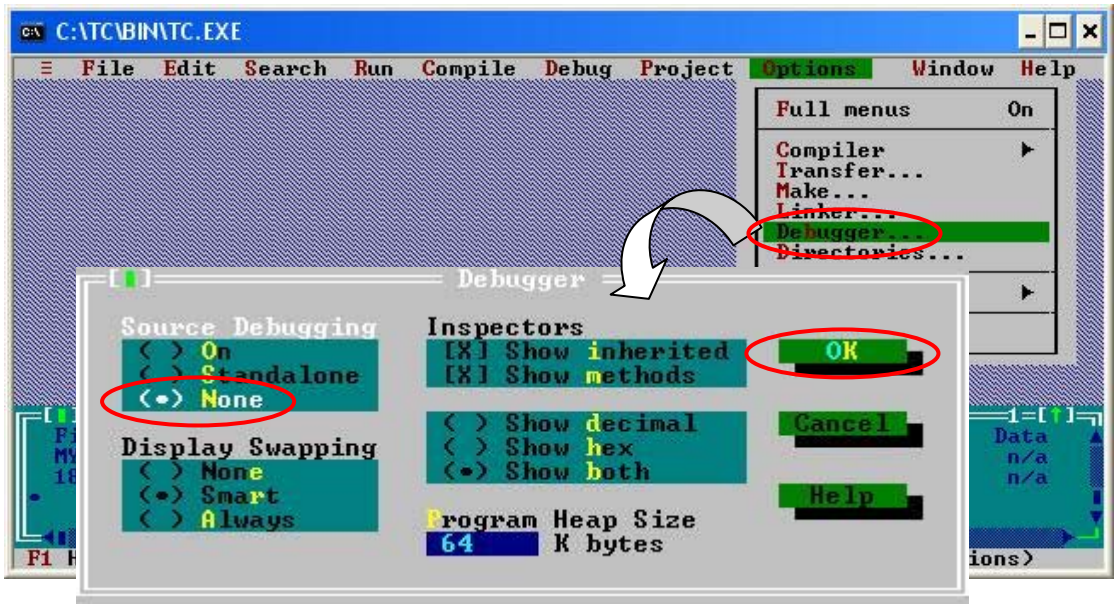
Step7: After finishing the Step6, the TC++1.01 window will look like as follows.



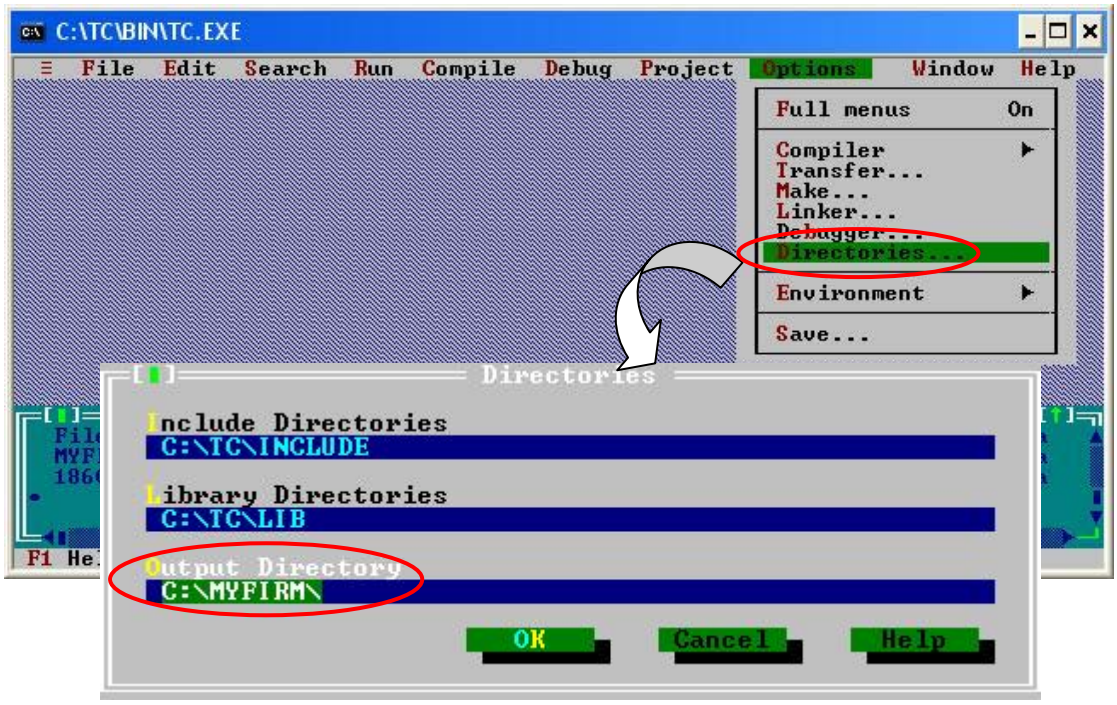
Step8: Click the “Options/Compiler/Code generation…” to set the compiler model to the large mode. Afterwards, click “More…” to set the “Floating point” and “Instruction Set” parameters, the Emulation and 80186 item will be used respectively. Then, click OK to save the configuration.



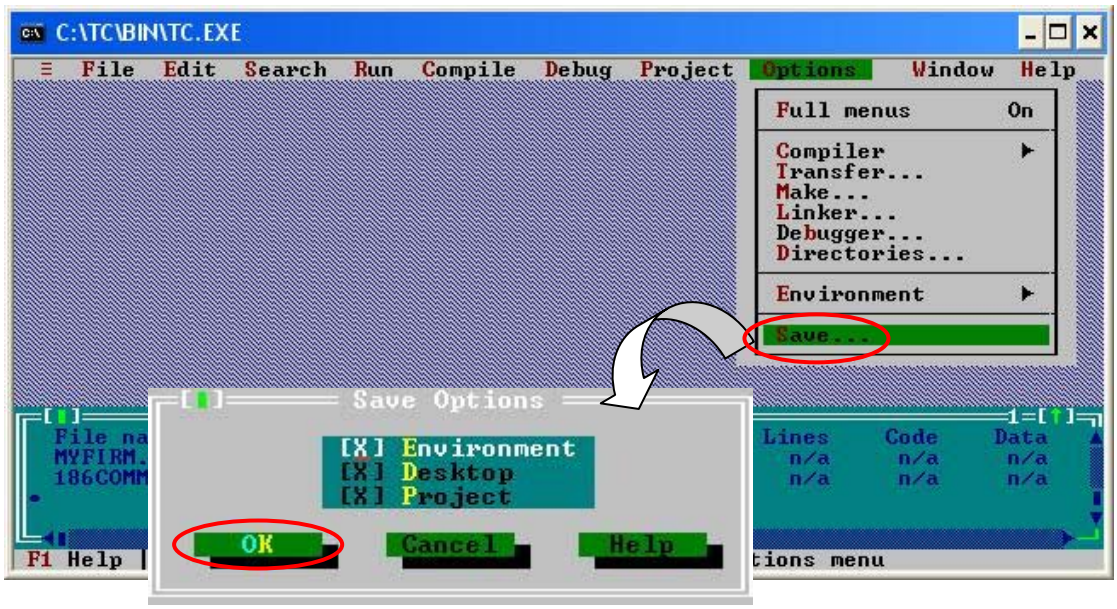
Step9: Click the “Option/Debugger...” to set the “Source Debugging” parameter. Here, select “None” for this parameter setting.



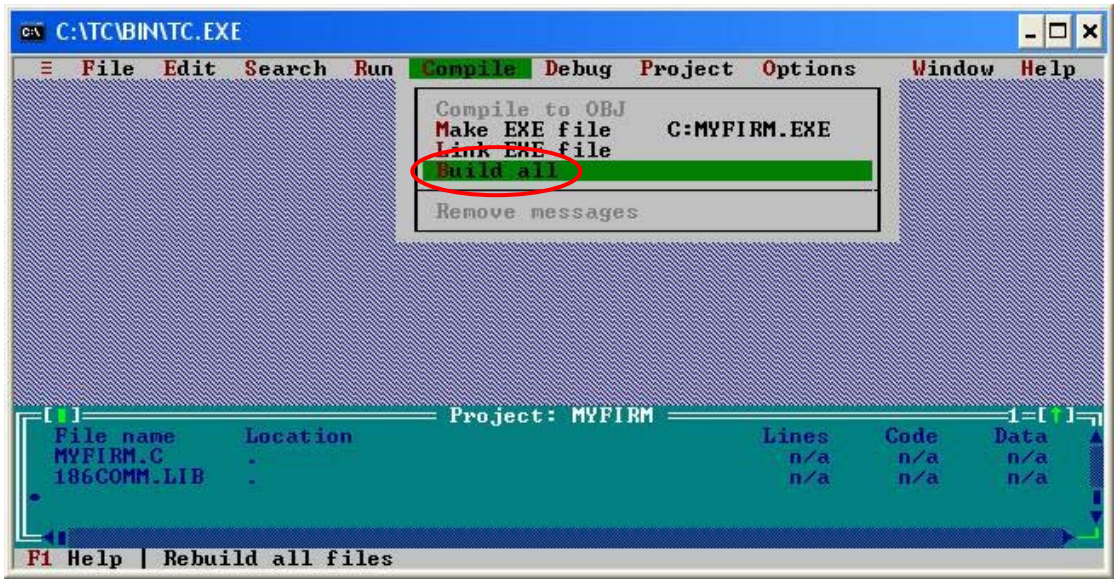
Step10: Click the “Option/Directories...” to set the “Output Directory” parameter.
Here, set the “C:\MyFirm” for the “Output Directory” parameter.



Step11: After finishing the parameters setting, click the “Options/save” to save this project.



Step12: After finishing the parameters setting, click the “Compile/build all” to produce the execution file. Users can find the execution file in the MyFirm folder. Its name is MyDemo.exe. The warning messages may occur during the compiling procedure because the INTT parameters of UserCANIrqFunc() and UserDPRAMIrqFunc() are not used. These warning will not have any affection to user-defined firmware.



Step15: Use ftp to copy the MyFirm.exe built before to the WinPAC/LinPAC. For iPAC, you need use the COM1 of iPAC and 7188xw.exe to download it.

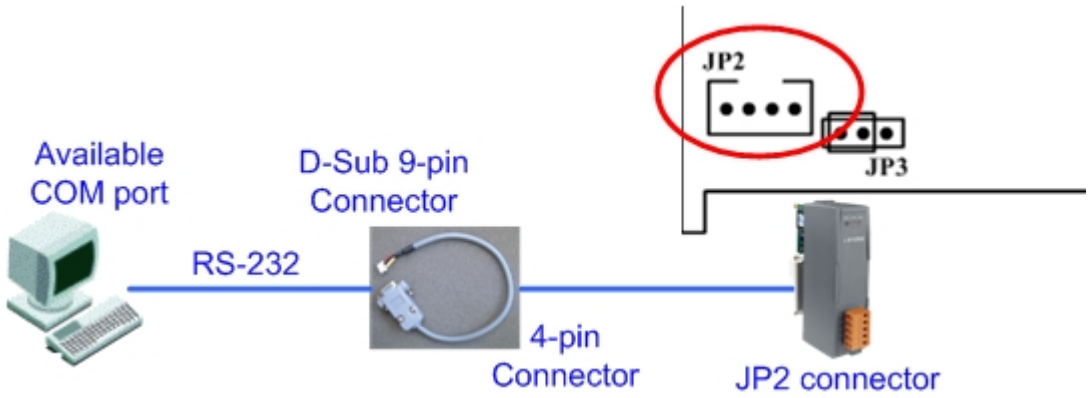
Step16: Use I8120W_Utility to download user-defined firmware to the I-8120W. Users can refer to section 3.4 for more details.

3.7 Debug Tools for User-defined Firmware Programming

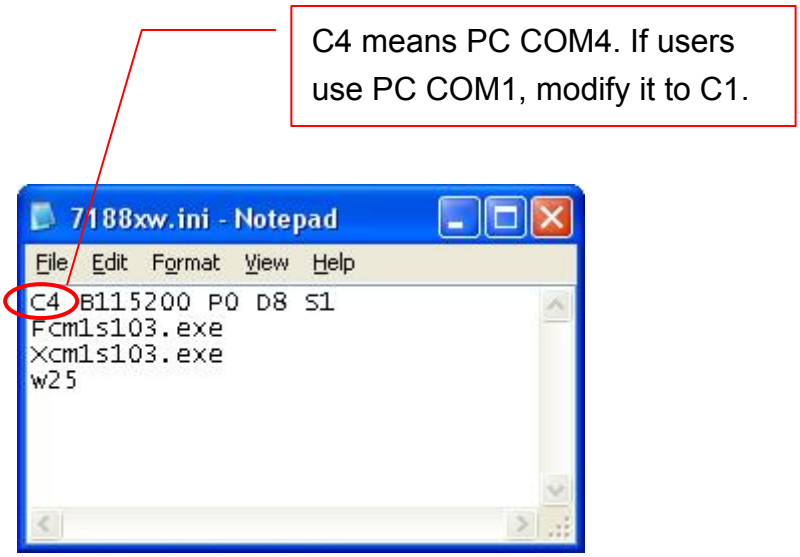
If users just apply default firmware for their application, this section can be ignored. This section introduces the debug methods when users deign their firmware. Basically, when users develop the user-defined firmware, the debug message can be put into the code section of user-defined firmware which may have bugs inside. Then, compile user-defined firmware, and download it into I-8120W. Owing to check the debug message, the bugs could be found. The following paragraph show how to use 7188xw.exe assisted with debug cable to debug users' firmware.

The firmware library provides two functions for applying. The function GetKbhit() allows users to received a character in firmware. This character is produced when users key something on 7188xw.exe. Then, users can use this feature to trigger some specified event for debugging. The function Print() allows users to send debug messages to 7188xw.exe. Afterwards, these debug messages will be put on the screen of 7188xw.exe.

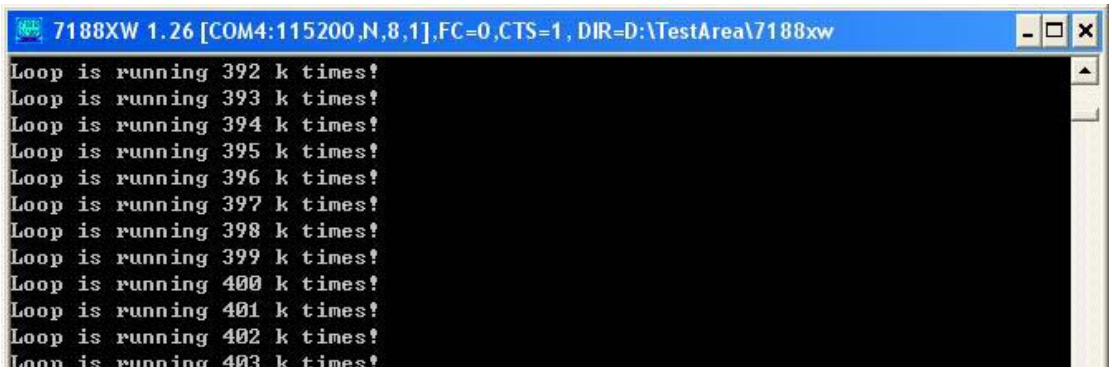
Before debugging your firmware, you need to prepare an debug cable. Plug the debug cable to the JP2 of I-8120W described in section 2.2. Connect an available PC COM port with the D-Sub 9-pin connector of debug cable. The architecture is shown as following figure.



Then, use Notepad.exe to modify the 7188xw.ini to set the number of specified PC COM port which is connecting with debug cable. The configuration screen is displayed as following figure. Finally, put this 7188xw.ini and 7188xw.exe in the same folder and execute the 7188xw.exe. Users can find 7188xw.ini and 7188xw.exe in the "Field Bus" CD. The path is CAN\SlotModule\I-8120W\Tools\PC\.



Then, any keyboard input to 7188xw.exe on the PC will be caught by user-defined firmware via GetKbhit() function. The debug messages sent by Print() function in firmware will also be displayed on the screen of 7188xw.exe.



4 APIs for Windows Application

In this chapter, the APIs for both default firmware and user-defined firmware are described. The content includes the APIs introductions, error code description and the simple method of troubleshooting. It is helpful to development users' application. The section 4.1 shows the list and information of all APIs supported by I-8120W. The section 4.2 shows the explication of the return codes of the API functions. It can help users to shoot their troubles when building an application.

4.1 API Definitions and Descriptions

All the functions provided by I-8120W are listed in the following table and the details for each function will be presented in the following sub-section.

Function definition	Note
WORD I8120_GetDllVersion(void)	○△
int I8120_AdujstDateTime(BYTE SlotNo)	○△
int I8120_Reset(BYTE SlotNo)	○△
int I8120_Init(BYTE SlotNo)	○△
int I8120_HardwareReset(BYTE SlotNo)	○△
int I8120_Check186Mode(BYTE SlotNo, BYTE *Mode)	○△
int I8120_Status(BYTE SlotNo, BYTE *bStatus)	○△
int I8120_AddCyclicTxMsg(BYTE SlotNo, BYTE Mode, DWORD MsgID, BYTE RTR, BYTE DataLen, BYTE *Data, DWORD TimePeriod, DWORD TransmitTimes, BYTE *Handle)	○△
int I8120_DeleteCyclicTxMsg(BYTE SlotNo, BYTE Handle)	○△
int I8120_EnableCyclicTxMsg(BYTE SlotNo, BYTE Handle)	○△
int I8120_DisableCyclicTxMsg(BYTE SlotNo, BYTE Handle)	○△
void I8120_OutputByte(BYTE SlotNo, WORD wOffset, BYTE bValue)	○△
int I8120_InputByte(BYTE SlotNo, WORD wOffset, BYTE *GetData)	○△
int I8120_IsTxTimeout(BYTE SlotNo, BYTE *Status)	○△
int I8120_SetSystemMsg(BYTE SlotNo, BYTE Mode)	○△
int I8120_EnableSJA1000(BYTE SlotNo)	○△

Function definition	Note
int I8120_DisableSJA1000(BYTE SlotNo)	○△
int I8120_RestoreI8120(BYTE SlotNo)	○△
int I8120_ClearSoftBuffer(BYTE SlotNo)	○
int I8120_ClearBufferStatus(BYTE SlotNo)	○
int I8120_ClearDataOverrun(BYTE SlotNo)	○
int I8120_Config(BYTE SlotNo, ConfigStruct *CanConfig)	○
int I8120_ConfigWithoutStruct(BYTE SlotNo, DWORD AccCode, DWORD AccMask, BYTE BaudRate, BYTE BT0, BYTE BT1)	○
int I8120_RxMsgCount(BYTE SlotNo)	○
int I8120_ReceiveMsg(BYTE SlotNo, PacketStruct *CanPacket)	○
int I8120_ReceiveWithoutStruct(BYTE SlotNo, BYTE *Mode, DWORD *MsgID, BYTE *RTR, BYTE *DataLen, BYTE *Data , DWORD *UpperTime , DWORD *LowerTime)	○
int I8120_SendMsg(BYTE SlotNo, PacketStruct *CanPacket)	○
int I8120_SendWithoutStruct(BYTE SlotNo, BYTE Mode, DWORD MsgID, BYTE RTR, BYTE DataLen, BYTE *Data)	○
int I8120_SJA1000Config(BYTE SlotNo, DWORD AccCode, BYTE BaudRate, BYTE BT0, BYTE BT1)	△
int I8120_DPRAMInttToI8120(BYTE SlotNo, BYTE Data)	△
int I8120_DPRAMWriteByte(BYTE SlotNo, WORD Address, BYTE Data)	△
int I8120_DPRAMWriteWord(BYTE SlotNo, WORD Address, WORD Data)	△
int I8120_DPRAMWriteDword(BYTE SlotNo, WORD Address, DWORD Data)	△
int I8120_DPRAMWriteMultiByte(BYTE SlotNo, WORD Address, BYTE *Data, WORD DataNum)	△
int I8120_DPRAMReadByte(BYTE SlotNo, WORD Address, BYTE *Data)	△
int I8120_DPRAMReadWord(BYTE SlotNo, WORD Address, WORD *Data)	△
int I8120_DPRAMReadDword(BYTE SlotNo, WORD Address, DWORD *Data)	△
int I8120_DPRAMReadMultiByte(BYTE SlotNo, WORD Address, BYTE *Data, WORD DataNum)	△
int I8120_DPRAMMemset(BYTE SlotNo, WORD Address, BYTE Data, WORD DataNum)	△
int I8120_ReceiveCmd(BYTE SlotNo, BYTE *Data, WORD *DataNum)	△
int I8120_SendCmd(BYTE SlotNo, BYTE *Data, WORD DataNum)	△

Function definition	Note
int I8120_InstallUserISR(BYTE SlotNo, void (*UserISR)(BYTE SlotNo, BYTE InttValue))	△
int I8120_RemoveUserISR(BYTE SlotNo)	△

Table 4.1 I-8120W Windows APIs List

Note: In table 3.1, the mark ○ and △ indicate the valid condition of API functions. The function marked by ○ or △ presents that this function is useful when the default firmware or user-defined firmware is inside the I8120W respectively. If users use default firmware, all of the functions marked by ○ could be applied. However, if users design their own firmware by using firmware library (refer to section 3.6), only the functions marked by △ is useful.

In order to make the descriptions more simplified and clear, the attributes for the input and output parameter of APIs are given as **[input]** and **[output]** respectively. They are described as follows.

Keyword	Set parameter by user before calling this function?	Get the data from this parameter after calling this function?
[input]	Yes	No
[output]	No	Yes

Table 4.2 Description of API parameter Hint

4.1.1 I8120_GetDllVersion

- **Description:**
  Obtain the version information of the library of I-8120W.

- **Syntax:**
  WORD I8120_GetDllVersion(void)

- **Parameter:**
  None

- **Return:**
  Function library version information. For example: If the value 200 is return, it means the library version is 2.00.

4.1.2 I8120_ AdujstDateTime

- **Description:**
  Adjust date and time of I-8120W by using the system time of WinPAC/LinPAC/iPAC.

- **Syntax:**
  int I8120_AdujstDateTime(BYTE SlotNo)

- **Parameter:**
  SlotNo: [input] I-8120W slot No. (0~7).

- **Return:**
  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.
  I8120_TimeOut: The I-8120W has no response.
  I8120_SetDateTimeFailure: Set date and time failure.

4.1.3 I8120_Reset

- **Description:**
  Reset the CAN controller, SJA1000, of the I-8120W.

- **Syntax:**
  int I8120_Reset(BYTE SlotNo)

- **Parameter:**
  SlotNo: [input] I-8120W slot No. (0~7).

- **Return:**
  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.
  I8120_TimeOut: The I-8120W has no response.
  I8120_ModeError: The I-8120W is in download mode, and can't be changed to firmware mode.

4.1.4 I8120_Init *<must be called once >*

- **Description:**
  Initiate the specific I-8120W. This function will be used when the application is at beginning. After calling this function, users must call the function I8120_HardwareReset to finish the initialization of the I-8120W.

- **Syntax:**
  int I8120_Init(BYTE SlotNo)

- **Parameter:**
  SlotNo: [input] I-8120W slot No. (0~7).

- **Return:**
  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

4.1.5 I8120_HardwareReset <must be called once >

- **Description:**
  Reset the I-8120W hardware, such as CAN controller, DPRAM, 186 CPU, …, and so forth.

- **Syntax:**
  int I8120_HardwareReset(BYTE SlotNo)

- **Parameter:**
  SlotNo: [input] I-8120W slot No. (0~7).

- **Return:**
  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.
  I8120_TimeOut: The I-8120W has no response.
  I8120_ModeError: The I-8120W is in download mode, and can't be changed to firmware mode.
  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.6 I8120_Check186Mode

- **Description:**
  Obtain the specified I-8120W if it is in download mode or in firmware mode.

- **Syntax:**
  int I8120_Check186Mode(BYTE SlotNo, BYTE *Mode)

- **Parameter:**
  SlotNo: [input] I-8120W slot No. (0~7).
  *Mode: [output] The address of a variable used to get the I-8120W mode. If this value is 0, it indicates that the I-8120W is in download mode. If 1, it is in firmware mode. When I-8120W is in download mode, it can only update the firmware and the firmware will not work at the same time. Users can use the function I8120_Reset to set the I-8120W into firmware mode.

- **Return:**
  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.
  I8120_TimeOut: The I-8120W has no response.
  I8120_ReplyError: The response of the I-8120W is not match with the expected value.
  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.7 I8120_Status

- **Description:**
  Obtain the status of the CAN controller for the specific I-8120W.

- **Syntax:**
  int I8120_Status(BYTE SlotNo, BYTE *bStatus)

- **Parameter:**
  SlotNo: [input] I-8120W slot No. (0~7).
  *bStatus: [output] The address of a variable is applied to get the status value of CAN controller.

Bit	NAME	VALUE	STATUS
bit 7	Bus Status	1	bus-off
		0	bus-on
bit 6	Error Status	1	error
		0	ok
bit 5	Transmit Status	1	transmit
		0	idle
bit 4	Receive Status	1	receive
		0	idle
bit 3	Transmission Complete Status	1	complete
		0	incomplete
bit 2	Transmit Buffer Status	1	release
		0	locked
bit 1	Data Overrun Status	1	overrun
		0	absent
bit 0	Receive Buffer Status	1	full/not empty
		0	empty

Table 4.3 Bit interpretation of the bStatus.

- **Return:**
  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.
  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.8 I8120_AddCyclicTxMsg

- **Description:**

  Add a cyclic transmission message into I8120W firmware. Afterwards, uses can enable or disable this cyclic transmission messages by using the function I8120_EnableCyclicTxMsg and I8120_DelectCyclicTxMsg. The maximum number of the cyclic transmission messages is 5. After adding a cyclic transmission message, the handle for this message will be returned. The less value of handle indicates the higher priority of this cyclic transmission message.

- **Syntax:**

  int I8120_AddCyclicTxMsg(BYTE SlotNo, BYTE Mode, DWORD MsgID,
  BYTE RTR, BYTE DataLen, BYTE *Data,
  DWORD TimePeriod,
  DWORD TransmitTimes, BYTE *Handle)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Mode: [input] 0 for 11-bit message ID, 1 for 29-bit message ID.

  MsgID: [input] CAN message ID.

  RTR: [input] Set remote-transmit-request is used or not. 0 is for useless, 1 is for useful.

  DataLen: [input] CAN message data length. The maximum value is 8.

  *Data: [input] The start address of the data buffer of a CAN message. The maximum space of *Data is 8 bytes.

  TimePeriod: [input] The time period of cyclic transmission. This parameter is formatted by 0.1ms. The minimum value is 5.

  TransmitTimes: [input] The numbers of CAN messages will be transmitted. After the I-8120W transmit all of the CAN messages which users decide the numbers of by using this parameter, I-8120W will disable this cyclic transmission message automatically. Users can enable this cyclic transmission message to require the I-8120W to send these CAN messages again by using the function I8120_EnableCyclicTxMsg. If this parameter is set to 0, the I-8120W will send CAN message cyclically and continuously after users enable this cyclic transmission message.

*Handle: [output] The address of a variable is used to get the handle of a cyclic transmission. When users want to enable or disable the specified cyclic transmission, this value must be needed.

- **Return:**
  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.
  I8120_TimeOut: The I-8120W has no response.
  I8120_SetCyclicMsgFailure: The cyclic transmission messages are over 5 messages or I-8120W replies erroneously.
  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.9 I8120_DeleteCyclicTxMsg

- **Description:**
  Remove the specified cyclic transmission message which is added by the function I8120_AddCyclicTxMsg.

- **Syntax:**
  int I8120_DeleteCyclicTxMsg(BYTE SlotNo, BYTE Handle)

- **Parameter:**
  SlotNo: [input] I-8120W slot No. (0~7).
  Handle: [input] The handle of cyclic transmission message which is obtained by the function I8120_AddCyclicTxMsg.

- **Return:**
  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.
  I8120_TimeOut: The I-8120W has no response.
  I8120_SetCyclicMsgFailure: The I-8120W replies erroneously.
  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.10 I8120_EnableCyclicTxMsg

- **Description:**

  Enable the cyclic transmission message which is added by the function I8120_AddCyclicTxMsg before. After enabling the specified cyclic transmission message, I-8120W will transmit the specified CAN message by configured time period.

- **Syntax:**

  int I8120_EnableCyclicTxMsg(BYTE SlotNo, BYTE Handle)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Handle: [input] The handle of cyclic transmission message which is obtained by the function I8120_AddCyclicTxMsg.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_TimeOut: The I-8120W has no response.

  I8120_SetCyclicMsgFailure: The I-8120W replies erroneously.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.11 I8120_DisableCyclicTxMsg

- **Description:**
  Disable the cyclic transmission message which is enabled by the function I8120_EnableCyclicTxMsg.

- **Syntax:**
  int I8120_DisableCyclicTxMsg(BYTE SlotNo, BYTE Handle)

- **Parameter:**
  SlotNo: [input] I-8120W slot No. (0~7).
  Handle: [input] The handle of cyclic transmission message which is obtained by the function I8120_AddCyclicTxMsg.

- **Return:**
  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.
  I8120_TimeOut: The I-8120W has no response.
  I8120_SetCyclicMsgFailure: The I-8120W replies erroneously.
  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.12 I8120_OutputByte

- **Description:**
  Write the data to the specified SJA1000 register of the I-8120W.

- **Syntax:**
  void I8120_OutputByte(BYTE SlotNo, WORD wOffset, BYTE bValue)

- **Parameter:**
  SlotNo: [input] I-8120W slot No. (0~7).
  wOffset: [input] The register address of SJA1000.
  bValue: [input] The value written to the specified register.

- **Return:**
  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.
  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.13 I8120_InputByte

- **Description:**
  Read the data from the specified SJA1000 register of the I-8120W.

- **Syntax:**
  int I8120_InputByte(BYTE SlotNo, WORD wOffset, BYTE *GetData)

- **Parameter:**
  SlotNo: [input] I-8120W slot No. (0~7).
  wOffset: [input] The register address of SJA1000.
  *GetData: [output] The address of a variable is used to get the data of the specific register of SJA1000.

- **Return:**
  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.
  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.14 I8120_IsTxTimeout

- **Description:**

  Use this function to check if the I-8120W finish the transmission of CAN message or not. When users call the function I8120_SendMsg or I8120_SendWithoutStruct, I-8120W firmware will put the CAN message into message buffer until the SJA1000 is available. When users use this function to check if the CAN message is transmitted or not, the I-8120W may not reply immediately until 1 second later. Generally, the transmission timeout of the I-8120W is due to wrong baud, broken line and loose connector. Therefore, when users want to use this function, it is recommended that put this function in a timer schedule to check if the transmission timeout of the I-8120W occurs or not.

- **Syntax:**

  int I8120_IsTxTimeout(BYTE SlotNo, BYTE *Status)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  *Status: [output] The address of a variable is used to obtain the transmission status of I-8230W. The value 0 means that the transmission status is normal. If the value is 1, the transmission status is timeout.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.15 I8120_SetSystemMsg

- **Description:**

  When the I-8120W boots up, the firmware of the I-8120W prints some information about firmware version to the debug tool description in section 3.7. If users don't want to see the system information, call this function once.

- **Syntax:**

  int I8120_SetSystemMsg(BYTE SlotNo, BYTE Mode)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Mode: [input] If the value is 0, the system information is disable. If the value is 1, the system information is enable.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.16 I8120_EnableSJA1000

- **Description:**

  This function will enable the transmission, reception, and interrupt of SJA1000. If the SJA1000 by using the function I8120_DisableSJA1000, calling this function can recover the SJA1000 to be enabled. Moreover, after using the function I8120_ SJA1000Config, users always call this function to enable the SJA1000 sequentially.

- **Syntax:**

  int I8120_EnableSJA1000(BYTE SlotNo)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

  I8120_ReplyError: The response of the I-8120W is not match with the expected value.

4.1.17 I8120_DisableSJA1000

- **Description:**

  Call this function will disable all functions of SJA1000, such as sending CAN message, receiving CAN message and interrupt. Afterwards, users can call the function I8120_EnableSJA1000 to recover the all functions of SJA1000.

- **Syntax:**

  int I8120_DisableSJA1000(BYTE SlotNo)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

  I8120_ReplyError: The response of the I-8120W is not match with the expected value.

4.1.18 I8120_RestoreI8120 *<must be called once >*

- **Description:**

  When users want to close the application, users must call this function to release the system resource and the interrupt function. If uses don't call this function before close application, the system may be crashed or be unstable.

- **Syntax:**

  int I8120_RestoreI8120(BYTE SlotNo)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_TimeOut: The I-8120W has no response.

  I8120_ModeError: The I-8120W is in download mode, and can't be changed to firmware mode.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.19 I8120_ClearSoftBuffer <For default firmware>

- **Description:**

  Clear the software buffer of the I-8120W. When users get the return code I8120_SoftBufferIsFull from the function I8120_SendWithoutStruct, I8120_SendMsg, I8120_ReceiveWithoutStuct or I8120_ReceiveMsg, this function may be needed.

- **Syntax:**

  int I8120_ClearSoftBuffer(BYTE SlotNo)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_SlotNotConfig: Call this function without calling the function I8120_Config or I8120_ConfigWithoutStruct before.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.20 I8120_ClearBufferStatus <For default firmware>

- **Description:**

  Use this function to clear the status of reception software buffer and transmission software buffer of the I-8120W. If the hardware buffer of SJA1000 is overflow, call this function will also reset the SJA1000.

- **Syntax:**

  int I8120_ClearBufferStatus(BYTE SlotNo)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_TimeOut: The I-8120W has no response.

  I8120_SlotNotConfig: Call this function without calling the function I8120_Config or I8120_ConfigWithoutStruct before.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.21 I8120_ClearDataOverrun <For default firmware>

- **Description:**

  Clear the data overrun status of SJA1000. When users use the function I8120_Status to get the status of SJA1000 and get the value 1, this function may be needed.

- **Syntax:**

  int I8120_ClearDataOverrun(BYTE SlotNo)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_SlotNotConfig: Call this function without calling the function I8120_Config or I8120_ConfigWithoutStruct before.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.22 I8120_Config <For default firmware>

- **Description:**

  Configure the baud, message filter of SJA1000. After calling this function, the I-8120W can start to send/receive CAN messages to/from the CAN network.

- **Syntax:**

  int I8120_Config(BYTE SlotNo, ConfigStruct *CanConfig)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  * CanConfig: [input] The address of a ConfigStruct structure variable used to configure the I-8120W. The ConfigStruct structure is defined as following:

```
typedef struct{
        BYTE AccCode[4];
        BYTE AccMask[4];
        BYTE BaudRate;
        BYTE BT0,BT1;
} ConfigStruct;
```

  AccCode[4]: Acceptance code of CAN controller.

  AccMask[4]: Acceptance mask of CAN controller.

  The AccCode is used for deciding what kind of ID the CAN controller will accept. The AccMask is used for deciding which bit of ID will need to check with AccCode. If the bit of AccMask is set to 0, it means that the bit in the same position of ID need to be checked, and that ID bit value needs to match the bit of AccCode in the same position. The regulations for 29-bit ID and 11-bit ID is shown in the table on the next page.

AccCode and AccMask	Bit Position	Filter Target
high byte of the high word	bit7~bit0	bit10 ~ bit3 of ID
low byte of the high word	bit7~bit5	bit2 ~ bit0 of ID
low byte of the high word	bit4	RTR
low byte of the high word	bit3~bit0	no use
high byte of the low word	bit7~bit0	bit7 ~ bit0 of 1st byte data
low byte of the low word	bit7~bit0	bit7 ~ bit0 of 2nd byte data

Table 4.4 AccCode and AccMask Definition For 11-bit ID

AccCode and AccMask	Bit Position	Filter Target
high byte of the high word	bit7~bit0	bit28~ bit21 of ID
low byte of the high word	bit7~bit0	bit20 ~ bit13 of ID
high byte of the low word	bit7~bit0	bit12 ~ bit5 of ID
low byte of the low word	bit7~bit3	bit4 ~ bit0 of ID
low byte of the low word	bit2	RTR
low byte of the low word	bit1~bit0	no use

Table 4.5 AccCode and AccMask Definition For 29-bit ID

For example (In 29 bit ID message):

		Array[0]	Array[1]	Array[2]	Array[3]	
AccCode	:	00h	00h	00h	A0h	
AccMask	:	FFh	FFh	FFh	1Fh	
ID bit		bit28~bit21	bit20~bit13	bit12~bit5	bit4~bit0	
ID Value	:	xxxx xxxx	xxxx xxxx	xxxx xxxx	101x x	will be accepted

(Note: The mark “x” means don’t care. And the mark “h” behind the value means hex format.)

BaudRate:

Value	Description
0	User-defined baud (BT0,BT1 are needed)
1	10 K bps
2	20 K bps
3	50 K bps
4	125 K bps
5	250 K bps
6	500 K bps
7	800 K bps
8	1000 K bps

Table 4.6 Relation Between BaudRate value and Baud

BT0, BT1: User-defined baud rate (used only if BaudRate=0). For example, set BT0=0x04 and BT1=0x1C, then baud setting for the CAN controller is 100Kbps. For more detailed baud setting, please refer to manual of SJA1000.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_TimeOut: The I-8120W has no response.

  I8120_InitError: The I-8120W replies configuration error.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.23 I8120_ConfigWithoutStruct <For default firmware>

- **Description:**

  This function is similar with the function I8120_Config. The difference is the input parameters of the function. This function uses no structure parameter so that it is easy to be applied in some program environment, such as VB.Net. Therefore, about the input parameters of this function please refer to the function I8120_Config function for the details.

- **Syntax:**

  int I8120_ConfigWithoutStruct(BYTE SlotNo, DWORD AccCode, DWORD AccMask, BYTE BaudRate, BYTE BT0, BYTE BT1)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).
  AccCode: [input] Acceptance code of CAN controller.
  AccMask: [input] Acceptance mask of CAN controller.
  BaudRate: [input] The baud indicator of CAN controller.
  BT0: [input] User-defined baud.
  BT1: [input] User-defined baud.

  For more information about these parameters, please refer to the section 3.2.28.

- **Return:**

  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.
  I8120_TimeOut: The I-8120W has no response.
  I8120_InitError: The I-8120W replies configuration error.
  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.24 I8120_RxMsgCount <For default firmware>

- **Description:**

  Obtain the number of CAN messages available in the reception software buffer of the WinPAC/LinPAC/iPAC.

- **Syntax:**

  int I8120_RxMsgCount(BYTE SlotNo, WORD *RxMsgCnt)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  *RxMsgCnt: [output] The address of a variable is used to get the numbers of the available CAN messages in the software buffer.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_SlotNotConfig: Call this function without calling the function I8120_Config or I8120_ConfigWithoutStruct before.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.25 I8120_ReceiveMsg <For default firmware>

- **Description:**

  Obtain the received messages from software buffer. Before using this function, the CAN controller must be configured by using the function I8120_Config or I8120_ConfigWithoutStruct.

- **Syntax:**

  int I8120_ReceiveMsg(BYTE SlotNo, PacketStruct *CanPacket)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  *CanPacket: [output] The address of a PacketStruct structure variable used to get a CAN message. The PacketStruct structure is defined as following:

```
typedef struct packet{
        LONGLONG MsgTimeStamps;
        BYTE    mode;
        DWORD id;
        BYTE    rtr;
        BYTE    len;
        BYTE    data[8];
} PacketStruct;
```

  MsgTimeStamps: This parameter will record the time when I-8120W got a CAN message. This is formatted by 0.1 ms. The time base of this value refers to the hardware clock of I-8120W. When the personal computer boots up, the hardware clock starts to count.

  mode: 0 for 11-bit message ID, 1 for 29-bit message ID.

  id: CAN message ID.

  rtr: 0 for remote-transmit-request format is not used, 1 for remote-transmit-request is used.

  len: Data length of a CAN message

  data[8]: data of a CAN message

- **Return:**

  I8120_NoError: OK

I8120_SlotNumberError: There is no I-8120W on the specific slot No.

I8120_SoftBufferIsEmpty: There is no CAN message in reception software buffer.

I8120_SoftBufferIsFull: Users can still get CAN message from the reception software buffer, but the software buffer is overflow.

I8120_TimeOut: The I-8120W has no response.

I8120_SlotNotConfig: Call this function without calling the function I8120_Config or I8120_ConfigWithoutStruct before.

I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.26 I8120_ReceiveWithoutStruct <For default firmware>

- **Description:**

  Obtain a received message from software buffer. This function is similar with the function I8120_ReceiveMsg. The difference is that this function doesn't use any structure parameter. It is easy to use in some program environment, such as VB.Net.

- **Syntax:**

```
int I8120_ReceiveWithoutStruct(BYTE SlotNo, BYTE *Mode,
                               DWORD *MsgID, BYTE *RTR,
                               BYTE *DataLen, BYTE *Data,
                               DWORD *UpperTime,
                               DWORD *LowerTime)
```

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  *Mode: [output] The address of a variable used to get the mode of a CAN message. If value is 0, the received CAN message is with 11-bit ID. The 29-bit ID of a CAN message will have value 1.

  *MsgID: [output] The address of a variable used to get the CAN message ID.

  *RTR: [output] The address of a variable used to obtain the status of this CAN message. 0 for remote-transmit-request format is not used, 1 for remote- transmit-request is used.

  *DataLen: [output] The address of a variable used to obtain the data length of a CAN message. The range of this value is from 0 to 8.

  *Data: [output] The start address of a buffer used to get the data of a CAN message. Users need to put an 8-byte element array in this filed.

  *UpperTime: [output] The address of a variable used to obtain the higher double-word of time stamp of a CAN message.

  *LowerTime: [output] The address of a variable used to obtain the lower double-word of time stamp of a CAN message. The unit of UpperTime and LowerTime are 0.1ms.

- **Return:**

  I8120_NoError: OK

I8120_SlotNumberError: There is no I-8120W on the specific slot No.

I8120_SoftBufferIsEmpty: There is no CAN message in reception software buffer.

I8120_SoftBufferIsFull: Users can still get CAN message from the reception software buffer, but the software buffer is overflow.

I8120_TimeOut: The I-8120W has no response.

I8120_SlotNotConfig: Call this function without calling the function I8120_Config or I8120_ConfigWithoutStruct before.

I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.27 I8120_SendMsg <For default firmware>

- **Description:**

  Send a CAN message to software transmission buffer. When the CAN bus is idle, this CAN message will be sent to CAN network. Note that if users make some mistakes of CAN bus wiring and configuration, the CAN messages may not be transmitted successfully. In this case, the messages sent by users will be put in the transmission buffer. Users can use the function I8120_IsTxTimeout or I8120_Status to check if any error happens.

- **Syntax:**

  int I8120_SendMsg(BYTE SlotNo, PacketStruct *CanPacket)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  *CanPacket: [input] The address of a PacketStruct structure variable used to describe the sent CAN message. About the definition of PacketStruct, please refer to the description of I8120_ReceiveMsg() function.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_TimeOut: The I-8120W has no response.

  I8120_SoftBufferIsFull: The transmission software buffer is overflow.

  I8120_SlotNotConfig: Call this function without calling the function I8120_Config or I8120_ConfigWithoutStruct before.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.28 I8120_SendWithoutStruct <For default firmware>

- **Description:**

  Send a CAN message to software transmission buffer. When the CAN bus is idle, this CAN message will be sent to CAN network. This function is similar with the function I8120_SendMsg. The difference is that this function doesn't use any structure parameter. It is easy to use in some program environment, such as VB.Net. Note that if users make some mistakes of CAN bus wiring and configuration, the CAN messages may not be transmitted successfully. In this case, the messages sent by users will be put in the transmission buffer. Users can use the function I8120_IsTxTimeout or I8120_Status to check if any error happens.

- **Syntax:**

  int I8120_SendWithoutStruct(BYTE SlotNo, BYTE Mode, DWORD MsgID, BYTE RTR, BYTE DataLen, BYTE *Data)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Mode: [input] 0 for 11-bit message ID, 1 for 29-bit message ID.

  MsgID: [input] CAN message ID.

  RTR: [input] 0 for remote-transmit-request format is not used, 1 for remote-transmit-request is used.

  DataLen: [input] Data length of a transmitted CAN message. The maximum value is 8.

  *Data: [input] The start address of a buffer is used to store the transmitted data of a CAN message.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_TimeOut: The I-8120W has no response.

  I8120_SoftBufferIsFull: The transmission software buffer is overflow.

  I8120_SlotNotConfig: Call this function without calling the function I8120_Config or I8120_ConfigWithoutStruct before.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.29 I8120_SJA1000Config *<For user-defined firmware>*

- **Description:**

  Configure the message filter and baud of SJA1000. About the input parameters of this function please refer to the function I8120_Config for the details. After using this function, users must use the function I8120_EnableSJA100 to enable the SJA1000 of I-8120W.

- **Syntax:**

  int I8120_SJA1000Config(BYTE SlotNo, DWORD AccCode,
  DWORD AccMask, BYTE BaudRate,
  BYTE BT0, BYTE BT1)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  AccCode: [input] Acceptance code of CAN controller.
  AccMask: [input] Acceptance mask of CAN controller.
  BT0: [input] User-defined baud.
  BT1: [input] User-defined baud.
  For the more information about these parameters, please refer to the section 3.2.28.

- **Return:**

  I8120_NoError: OK
  I8120_SlotNumberError: There is no I-8120W on the specific slot No.
  I8120_TimeOut: The I-8120W has no response.
  I8120_InitError: The I-8120W replies configuration error.
  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.30 I8120_DPRAMInttToI8120 <For user-defined firmware>

- **Description:**

  Send an interrupt signal to I-8120W. This interrupt signal will pass to the user-defined firmware. Therefore, users can do something for it. Be careful that too many interrupt signals at a short time will affect the normal procedure of the user-defined firmware.

- **Syntax:**

  int I8120_DPRAMInttToI8120(BYTE SlotNo, BYTE Data)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Data: [input] Interrupt indicator. The range is from 0x00 to 0xdf. Users can define their own interrupt indicator and do some specified thing for it in user-defined firmware.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_DpramOverRange: The data of input parameter is over 0xdf.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.31 I8120_DPRAMWriteByte <For user-defined firmware>

- **Description:**

  Write one byte data into the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int I8120_DPRAMWriteByte(BYTE SlotNo,
  WORD Address, BYTE Data)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Address: [input] The specified address of DPRAM where users want to write data.

  Data: [input] The byte data written to the DPRAM of I-8120W.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_DpramOverRange: The Address of input parameter is over 6999.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.32 I8120_DPRAMWriteWord *<For user-defined firmware>*

- **Description:**

  Write one word data into the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int I8120_DPRAMWriteWord(BYTE SlotNo, WORD Address,
  WORD Data)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Address: [input] The specified address of DPRAM where users want to write data.

  Data: [input] The word data written to the DPRAM of I-8120W.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_DpramOverRange: The Address of input parameter is over 6998.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.33 I8120_DPRAMWriteDword <For user-defined firmware>

- **Description:**

  Write one double-word data into the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int I8120_DPRAMWriteDword(BYTE SlotNo,
  WORD Address, DWORD Data)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Address: [input] The specified address of DPRAM where users want to write data.

  Data: [input] The double-word data written to the DPRAM of I-8120W.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_DpramOverRange: The Address of input parameter is over 6996.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.34 I8120_DPRAMWriteMultiByte <For user-defined firmware>

- **Description:**

  Write multi-byte data into specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int I8120_DPRAMWriteMultiByte(BYTE SlotNo, WORD Address,
  BYTE *Data, WORD DataNum)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Address: [input] The specified start address of DPRAM where users want to write data.

  *Data: [input] The start address of a byte array written to the DPRAM of I-8120W.

  DataNum: [input] The byte number of an data array written to the DPRAM of I-8120W.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_DpramOverRange: The sum of Address and DataNum of input parameters is over 6999.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.35 I8120_DPRAMReadByte <For user-defined firmware>

- **Description:**

  Read one byte data from the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int I8120_DPRAMReadByte(BYTE SlotNo, WORD Address,
  BYTE *Data)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Address: [input] The specified address of DPRAM where users want to read data.

  *Data: [output] The address of a variable used to receive the data obtained by the I8120_DPRAMReadByte.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_DpramOverRange: The Address of input parameter is over 6999.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.36 I8120_DPRAMReadWord <For user-defined firmware>

- **Description:**

  Read one word data from the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int I8120_DPRAMReadWord(BYTE SlotNo, WORD Address,
  WORD *Data)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Address: [input] The specified address of DPRAM where users want to read data.

  *Data: [output] The address of a variable applied to receive the data obtained by the I8120_DPRAMReadWord.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_DpramOverRange: The Address of input parameter is over 6998.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.37 I8120_DPRAMReadDword <For user-defined firmware>

- **Description:**

  Read one double-word data from the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int I8120_DPRAMWriteDword(BYTE SlotNo, WORD Address,
  DWORD *Data)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Address: [input] The specified address of DPRAM where users want to write data.

  *Data: [output] The address of a variable applied to receive the data obtained by the function I8120_DPRAMReadDword.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_DpramOverRange: The Address of input parameter is over 6996.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.38 I8120_DPRAMReadMultiByte <For user-defined firmware>

- **Description:**

  Read multi-byte data into the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int I8120_DPRAMReadMultiByte(BYTE SlotNo, WORD Address,
  BYTE *Data, WORD DataNum)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Address: [input] The specified start address of DPRAM where users read to write data.

  *Data: [output] The start address of a byte array applied to receive the DPRAM data.

  DataNum: [input] The byte numbers which users will want to read from the DPRAM of I-8120W.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_DpramOverRange: The sum of Address and DataNum of input parameters is over 6999.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.39 I8120_DPRAMMemset <For user-defined firmware>

- **Description:**

  Set multi-byte DPRAM data to be the specified value. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int I8120_DPRAMMemset(BYTE SlotNo, WORD Address,
  BYTE Data, WORD DataNum)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  Address: [input] The specified start address of DPRAM where users want to write data.

  Data: [input] The data written to DPRAM of I-8120W.

  DataNum: [input] The byte numbers which users will want to write to DPRAM of I-8120W.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_DpramOverRange: The sum of Address and DataNum of input parameters is over 6999.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.40 I8120_ReceiveCmd <For user-defined firmware>

- **Description:**

  Use this function to receive the command transmitted from the user-defined firmware. When users use the function DPRAMSendCmd to a send command in the user-defined firmware, call this function to receive the command from the user-defined firmware. If users do not receive the command until another command is given from the user-defined firmware, the former one will be covered by the latter one. About the function DPRAMSendCmd, please refer to 5.1.17 for more information.

- **Syntax:**

  int I8120_ReceiveCmd(BYTE SlotNo, BYTE *Data, WORD *DataNum)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  *Data: [output] The start address of a byte array applied to receive the command from DPRAM of I-8120W.

  *DataNum: [output] The address of a variable applied to receive the command length.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_NoDpramCmd: There is no command transmitted from user-defined firmware.

  I8120_DpramOverRange: The command length is over 512 bytes.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.41 I8120_SendCmd <For user-defined firmware>

- **Description:**

  Call this function to send the command to user-defined firmware. The maximum command length is 512 bytes. Afterwards, users can use the function DPRAMReceiveCmd of firmware library to get this command. About the function DPRAMReceiveCmd, please refer to section 5.1.16 for more information.

- **Syntax:**

  int I8120_SendCmd(BYTE SlotNo, BYTE *Data, WORD DataNum)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  *Data: [input] The start address of a byte array of a sent command.

  DataNum: [input] The word value indicates how many bytes users will send to user-defined firmware.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_DpramOverRange: The command length is over 512 bytes.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.42 I8120_InstallUserISR *<For user-defined firmware>*

- **Description:**

  Using this function can allow users to apply ISR (interrupt service routine). When users put their ISR into this function, all of interrupt signals defined by users in user-defined firmware will trigger the users' ISR. Besides, the interrupt signal, CAN_COMM_CMD_FROM_I8120, defined in "I8120.h" will also pass to users' ISR when the WinPAC/LinPAC/iPAC get a DPRAM command from the user-defined firmware.

- **Syntax:**

  int I8120_InstallUserISR(BYTE SlotNo,
  void (*UserISR)(BYTE SlotNo, BYTE InttValue))

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

  (*UserISR)(BYTE SlotNo, BYTE InttValue)): [input] The pointer which points a function with format "void XXX(Byte SlotNo, Byte InttValue)". The XXX is the function name of users' ISR. The parameter, SlotNo, indicates the number of the board which produces an interrupt signal. The parameter, InttValue, is the interrupt indicator which may be the value CAN_COMM_CMD_FROM_I8120 or be the interrupt indicator transmitted from user-defined firmware.

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_DpramOverRange: The command length is over 512 bytes.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.1.43 I8120_RemoveUserISR <For user-defined firmware>

- **Description:**

  When users don't need the ISR function, call this function to remove users ISR.

- **Syntax:**

  int I8120_RemoveUserISR(BYTE SlotNo)

- **Parameter:**

  SlotNo: [input] I-8120W slot No. (0~7).

- **Return:**

  I8120_NoError: OK

  I8120_SlotNumberError: There is no I-8120W on the specific slot No.

  I8120_SlotNotInit: Call this function without calling the function I8120_Init before.

4.2 APIs Return Codes Troubleshooting

Return Code	Error ID (Error Description)	Troubleshooting
0	I8120_NoError	OK
3	I8120_SlotNumberError	1. Set the SlotNo parameter of function to match the DIP switch No.. 2. Each I-8120W, PISO-DNM- D/T, or PISO-CPM-D/T has unique DIP switch No.. 3. Unplug the I-8120W, and plug it again and turn on your PC until find it in the list of hardware management of Windows.
7	I8120_InitError	1. Retry the function again. 2. Call the function I8120_Init() and configure I-8120W again.
21	I8120_SoftBufferIsEmpty	1. Wait for a while and call the function again.
22	I8120_SoftBufferIsFull	1. Use I8120_ClearBufferStatus() to clear the status of buffer overflow. 2. Reduce the bus loading of CAN network
23	I8120_TimeOut	1. Wait for a while and call the function again. 2. Call the function I8120_Init() and configure I-8120W again. 3. Update default firmware again by using Utility if default firmware is used. 4. Confirm if user-defined firmware is in I-8120W or not by using Utility
24	I8120_SetCyclicMsgFailure	1. Check if users already use 5 the cyclic messages. 2. Call the function I8120_Init() and configure I-8120W again.
25	I8120_DpramOverRange	1. Check the Address or DataNum parameters of function if each of them or the sum of them exceeds 6999.
26	I8120_NoDpramCmd	1. Wait for a while and call the function again.
27	I8120_ModeError	1. Update the default firmware again by Utility tool if it is used.
30	I8120_NoFileInside	1. Update the default firmware again by using Utility if it is used. 2. Confirm if user-defined firmware is in I-8120W or not by using Utility
31	I8120_DownloadFailure	1. Close Utility and try to update the firmware one minute later. 2. Call your distributor to solve this problem
32	I8120_EEPROMDamage	1. Call your distributor to solve this problem
33	I8120_NotEnoughSpace	1. The file size of the user-defined firmware is too large to put it into the I-8120W.

Return Code	Error ID (Error Description)	Troubleshooting
34	I8120_StillDownloading	1. Close Utility and try to update firmware one minute later. 2. Call your distributor to solve this problem
35	I8120_BoardModeError	1. Close Utility and try to update the firmware one minute later.
36	I8120_SetDateTimeFailure	1. Call your distributor to solve this problem.
40	I8120_SlotNotConfig	1. Call the function I8120_Config or I8120_ConfigWithoutStruct before you call the function which gives you this return code.
41	I8120_SlotNotInit	1. Call the function I8120_Init at the start of application.
42	I8120_ReplyError	1. Call your distributor to solve this problem.

Table 4.7 Return Code Troubleshooting

Note: If users' problem can't be fixed after following the recommended methods. Please contact your distributor or email to service@icpdas.com to solve the problem.

5 Functions of Firmware Library

If the default firmware is used, users do not need to read this chapter. This chapter introduces all the functions provided by firmware library, 186COMM.lib. The content includes the description and list of functions of 186COMM.lib, error code description, and simple method of troubleshooting. It is helpful to build the user-defined firmware. The section 5.1 shows the list and information of all functions supported by 186COMM.lib. The section 5.2 is the basic troubleshooting when users apply the functions of 186COMM.lib and get an unexpected return code.

5.1 Firmware Library Definitions and Descriptions

When users want to design their own firmware, the functions of firmware library are needed. In order to reduce the development cycle, the firmware library, 186COMM.lib, provides 4 callback functions. If users want to do some initial job, put the program into the function UserInitFunc. Users' normal procedure can be put in the function UserLoopFunc. The firmware library will execute this callback function as soon as possible. If users would like to process some interrupt signal from DPRAM or CAN controller, use the callback functions UserDPRAMIrqFunc and UserCANIrqFunc() to do that. These 4 callback functions must be applied in users' .c file even users don't want to use them. The architecture is show as figure 5.1.

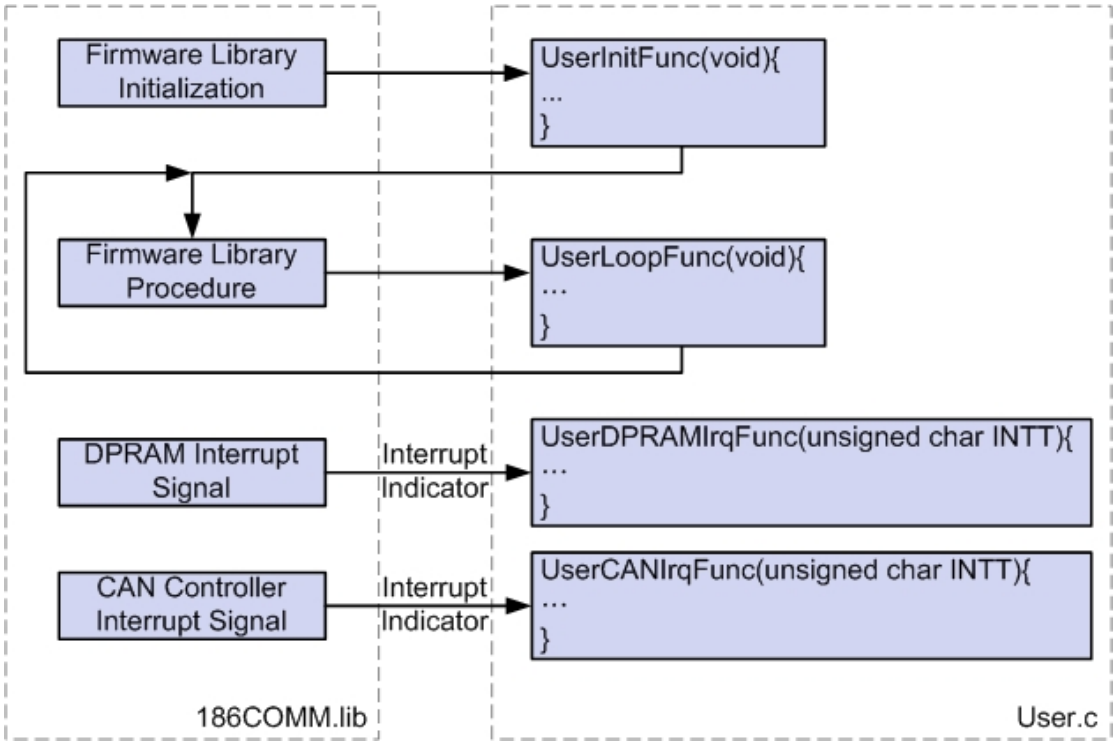


Figure 5.1 Firmware Library Operation Architecture

Besides, 186COMM.lib also supports some functions for handling the hardware of I-8120W, such as DPRAM access functions, EEPROM access functions, NVRAM access functions, LED control functions, real time clock access function, timer functions, debug functions, and CAN bus access functions. All the functions are listed in the table 5.1 and detailed information for every function is presented in the following sub-section.

Function definition
void L1Off(void)
void L1On(void)
void L2Off(void)
void L2On(void)
void DPRAMInttToHost(char InttValue)
void UserDPRAMIrqFunc(unsigned char INTT)
int DPRAMWriteByte(unsigned int Address, unsigned char Data)
int DPRAMWriteWord(unsigned int Address, unsigned int Data)
int DPRAMWriteDword(unsigned int Address, unsigned long Data)
int DPRAMWriteMultiByte(unsigned int Address, char *Data, unsigned int DataNum)
int DPRAMReadByte(unsigned int Address, unsigned char *Data)

Function definition
int DPRAMReadWord(unsigned int Address, unsigned int *Data)
int DPRAMReadDword(unsigned int Address, unsigned long *Data)
int DPRAMReadMultiByte(unsigned int Address, char *Data, unsigned int DataNum)
int DPRAMMemset(unsigned int Address, char data, unsigned int DataNum)
int DPRAMReceiveCmd(char *Data, unsigned int *DataNum)
int DPRAMSendCmd(char *Data, unsigned int DataNum)
int GetKbhit(void)
int Print(const char *fmt, ...)
void GetTime(int *hour, int *minute, int *sec)
int SetTime(int hour, int minute, int sec)
void GetDate(int *year, int *month, int *day)
int SetDate(int year, int month, int day)
int GetWeekDay(void)
int ReadNVRAM(int Address)
int WriteNVRAM(int Address, int data)
unsigned long GetTimeTicks100us(void)
long GetTimeTicks(void)
void DelayMs(unsigned int DelayTime_ms)
void CM100_InstallUserTimer(void (*Fun)(void))
void T_StopWatchStart(STOPWATCH *sw)
unsigned long T_StopWatchGetTime(STOPWATCH *sw)
void T_StopWatchPause(STOPWATCH *sw)
void T_StopWatchContinue(STOPWATCH *sw)
void T_CountDownTimerStart(COUNTDOWNTIMER *cdt, unsigned long timems)
void T_CountDownTimerPause(COUNTDOWNTIMER *cdt)
void T_CountDownTimerContinue(COUNTDOWNTIMER *cdt)
int T_CountDownTimerIsTimeUp(COUNTDOWNTIMER *cdt)
unsigned long T_CountDownTimerGetTimeLeft(COUNTDOWNTIMER *cdt)
int EEPROMReadByte(unsigned int Block, unsigned int Address, unsigned char *Data)
int EEPROMReadMultiByte(unsigned int Block, unsigned int Address, char *Data, unsigned int DataNum)
int EEPROMWriteByte(unsigned int Block, unsigned int Address, unsigned char Data)
int EEPROMWriteMultiByte(unsigned int Block, unsigned int Address, char *Data, unsigned int DataNum)
void UserCANIrqFunc(unsigned char INTT)
void SJA1000HardwareReset(void)
int SetCANBaud(unsigned long Baud, char BT0, char BT1)

Function definition
void GetCANBaud(unsigned long *Baud, char *BT0, char *BT1)
int SetCANMask(long AccCode, long AccMask)
void GetCANMask(long *AccCode, long *AccMask)
int CANConfig(unsigned long Baud, char BT0, char BT1, long AccMask, long AccCode)
void EnableSJA1000(void)
void DisableSJA1000(void)
int GetCANStatus(void)
void ClearDataOverrunStatus(void)
int SendCANMsg(char Mode, unsigned long MsgID, char RTR, char DataLen, char *Data)
void ClearTxSoftBuffer(void)
int GetCANMsg(char *Mode, unsigned long *MsgID, char *RTR, char *DataLen, char *Data, unsigned long *UpperTime, unsigned long *LowerTime)
void ClearRxSoftBuffer(void)
int RxMsgCount(void)
int AddCyclicTxMsg(char Mode, unsigned long MsgID, char RTR, char DataLen, char *Data, unsigned long TimePeriod, unsigned long TransmitTimes, unsigned char *Handle)
int DeleteCyclicTxMsg(unsigned char Handle)
int EnableCyclicTxMsg(unsigned char Handle)
int DisableCyclicTxMsg(unsigned char Handle)
void ResetCyclicTxBuf(void)
void SystemHardwareReset(void)
void SystemInit(void)
int GetLibVer(void)
void RefreshWDT(void)
void UserInitFunc(void)
void UserLoopFunc(void)

Table 5.1 Functions List of Firmware Library For User-defined Firmware

5.1.1 L1Off

- **Description:**
  Turn off the yellow LED of I-8120W.

- **Syntax:**
  void L1Off(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.2 L1On

- **Description:**
  Turn on the yellow LED of I-8120W.

- **Syntax:**
  void L1On(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.3 L2Off

- **Description:**
  Turn off the green LED of I-8120W.

- **Syntax:**
  void L2Off(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.4 L2On

- **Description:**
  Turn on the green LED of I-8120W.

- **Syntax:**
  void L1Off(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.5 DPRAMInttToHost

- **Description:**

  Call this function to signal the users' applications an interrupt. When users' applications receive the interrupt signal from the user-defined firmware, check the value of interrupt indicator to know the meaning of this interrupt. Therefore, the user-defined firmware can communicate with the Windows applications by the definitions of interrupt indicators. Because of the interrupt mechanism, too many calls of this function will increase host CPU loading and disturb the normal procedure of users' applications on WinPAC/LinPAC/iPAC.

- **Syntax:**

  void DPRAMInttToHost(char InttValue)

- **Parameter:**

  InttValue: [input] The interrupt indicator sent to users' Windows application. The range is from 0x00 ~ 0xdf.

- **Return:**

  None

5.1.6 UserDPRAMIrqFunc *<must be called once >*

- **Description:**

  This is a callback function, and must be call once in user-defined firmware. When firmware library receives an interrupt signal from users' Windows applications, it will pass the interrupt indicator from users' Windows applications to this function. Users can have some proper procedures in this function to process each interrupt indicator. It is not allowed to put an infinite loop in to this function, and users must keep the program of this function as short as possible.

- **Syntax:**

  void UserDPRAMIrqFunc(unsigned char INTT)

- **Parameter:**

  INTT: [input] The interrupt indicator from users' Windows application.

- **Return:**

  None

5.1.7 DPRAMWriteByte

- **Description:**
  Write one byte data into the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**
  int DPRAMWriteByte(unsigned int Address, unsigned char Data)

- **Parameter:**
  Address: [input] The specified address of DPRAM where users want to write data.
  Data: [input] The byte data written to the DPRAM of I-8120W.

- **Return:**
  _NO_ERR: OK
  _DPRAM_OVER_RANGE: The Address of input parameter is over 6999.

5.1.8 DPRAMWriteWord

- **Description:**
  Write one word data into the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**
  int DPRAMWriteWord(unsigned int Address, unsigned int Data)

- **Parameter:**
  Address: [input] The specified address of DPRAM where users want to write data.
  Data: [input] The word data written to the DPRAM of I-8120W.

- **Return:**
  _NO_ERR: OK
  _DPRAM_OVER_RANGE: The Address of input parameter is over 6998.

5.1.9 DPRAMWriteDword

- **Description:**

  Write one double-word data into the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int DPRAMWriteDword(unsigned int Address, unsigned long Data)

- **Parameter:**

  Address: [input] The specified address of DPRAM where users want to write data.

  Data: [input] The double-word data written to the DPRAM of I-8120W.

- **Return:**

  _NO_ERR: OK

  _DPRAM_OVER_RANGE: The Address of input parameter is over 6996.

5.1.10 DPRAMWriteMultiByte

- **Description:**

  Write multi-byte data into the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int DPRAMWriteMultiByte(unsigned int Address, char *Data,
  unsigned int DataNum)

- **Parameter:**

  Address: [input] The specified start address of DPRAM where users want to write data.

  *Data: [input] The start address of a byte array written to the DPRAM of I-8120W.

  DataNum: [input] The byte numbers of an data array written to the DPRAM of I-8120W.

- **Return:**

  _NO_ERR: OK

  _DPRAM_OVER_RANGE: The sum of Address and DataNum of input parameters is over 6999.

5.1.11 DPRAMReadByte

- **Description:**
  Read one byte data from specified the address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**
  int DPRAMReadByte(unsigned int Address, unsigned char *Data)

- **Parameter:**
  Address: [input] The specified address of DPRAM where users want to read data.
  *Data: [output] The address of a variable used to receive the data obtained by the function DPRAMReadByte.

- **Return:**
  _NO_ERR: OK
  _DPRAM_OVER_RANGE: The Address of input parameter is over 6999.

5.1.12 DPRAMReadWord

- **Description:**
  Read one word data from the specified address of DPRAM. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**
  int DPRAMReadWord(unsigned int Address, unsigned int *Data)

- **Parameter:**
  Address: [input] The specified address of DPRAM where users want to read data.
  *Data: [output] The address of a variable applied to receive the data obtained by the function DPRAMReadWord.

- **Return:**
  _NO_ERR: OK
  _DPRAM_OVER_RANGE: The Address of input parameter is over 6998.

5.1.13 DPRAMReadDword

- **Description:**

  Read one double-word data from the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int DPRAMReadDword(unsigned int Address, unsigned long *Data)

- **Parameter:**

  Address: [input] The specified address of DPRAM where users want to read data.

  *Data: [output] The address of a variable applied to receive the data obtained by the function DPRAMReadDword.

- **Return:**

  _NO_ERR: OK

  _DPRAM_OVER_RANGE: The Address of input parameter is over 6996.

5.1.14 DPRAMReadMultiByte

- **Description:**

  Write the multi-byte data into the specified address of DPRAM of I-8120W. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int DPRAMReadMultiByte(unsigned int Address, char *Data,
  unsigned int DataNum)

- **Parameter:**

  Address: [input] The specified start address of DPRAM where users want to read data.

  *Data: [output] The start address of a byte array applied to receive the data from DPRAM of I-8120W.

  DataNum: [input] The byte numbers which users will want to read from the DPRAM of I-8120W.

- **Return:**

  _NO_ERR: OK

  _DPRAM_OVER_RANGE: The sum of Address and DataNum of input parameters is over 6999.

5.1.15 DPRAMMemset

- **Description:**

  Set the multi-byte DPRAM data to be the specified value. The DPRAM space which can be applied is from address 0 to 6999.

- **Syntax:**

  int DPRAMMemset(unsigned int Address, char data,
  unsigned int DataNum)

- **Parameter:**

  Address: [input] The specified start address of DPRAM where users want to write data.

  Data: [input] The data written to DPRAM of I-8120W.

  DataNum: [input] The byte numbers which users will want to write to DPRAM of I-8120W.

- **Return:**

  _NO_ERR: OK

  _DPRAM_OVER_RANGE: The sum of Address and DataNum of input parameters is over 6999.

5.1.16 DPRAMReceiveCmd

- **Description:**

  Use this function to receive the command transmitted from the windows applications. When users use the function I8120_SendCmd to a send command in users' application of WinPAC/LinPAC/iPAC, call this function to receive the command which comes from the application. If users do not receive the command until another command is given from users' application, the former one will be covered by the latter one. About the function I8120_SendCmd, please refer to 4.1.41 for more information.

- **Syntax:**

  int DPRAMReceiveCmd(char *Data, unsigned int *DataNum)

- **Parameter:**

  *Data: [output] The start address of a byte array is applied to receive the command from DPRAM of I-8120W.

  *DataNum: [output] The address of a variable is applied to receive the command length.

- **Return:**

  _NO_ERR: OK

  _NO_DPRAM_CMD: There is no command transmitted from user-defined firmware.

  _DPRAM_OVER_RANGE: The command length is over 512 bytes.

5.1.17 DPRAMSendCmd

- **Description:**

  Call this function to send the command to users' applications of WinPAC/LinPAC/iPAC. The maximum command length is 512 bytes. Afterwards, users can use the function I8120_ReceiveCmd to get this command. About the function I8120_ReceiveCmd, please refer to section 4.1.40 for more information. The maximum command length can't exceed to 512 bytes.

- **Syntax:**

  int DPRAMSendCmd(char *Data, unsigned int DataNum)

- **Parameter:**

  *Data: [input] The start address of a byte array of a sent command.

  DataNum: [input] The word value indicates how many bytes users will send to the user-defined firmware.

- **Return:**

  _NO_ERR: OK

  _DPRAM_OVER_RANGE: The command length is over 512 bytes.

5.1.18 GetKbhit *<assist with debug cable and 7188xw.exe>*

- **Description:**

  This function is used for debugging of the user-defined firmware. Call this function to get a character keyed from keyboard. The function GetKbhit is similar with standard C function “getch”. When users connect the debug port of the I-8120W with the available RS-232 COM port of PC via the debug cable shown in section 2.2, execute the 7188xw.exe Windows program. Then, a character keyed from keyboard will be caught by this function.

- **Syntax:**

  int GetKbhit(void)

- **Parameter:**

  None

- **Return:**

  The return code is the received character from keyboard input when 7188xw.exe is executed and focused.

5.1.19 Print *<assist with debug cable and 7188xw.exe>*

- **Description:**

  This function is used for debugging of the user-defined firmware. Call this function to send the debug information to the 7188xw.exe. The function Print is similar with standard C function “printf”. When users connect the debug port of I-8120W with the available RS-232 COM port of PC via the debug cable shown in section 2.2, execute the 7188xw.exe Windows program. Then, the debug information sent by this function will be put on the screen of the 7188xe.wxe.

- **Syntax:**

  int Print(const char *fmt, ...)

- **Parameter:**

  * fmt: [input] The data format of keyboard input. Please refer to standard C function printf() to know how to use this parameters.

- **Return:**

  If it is successful, the return code is a non-zero value except the value of EOF (defined by standard C/C++ language).

5.1.20 GetTime

- **Description:**

  Use this function to get the current time from real time clock.

- **Syntax:**

  void GetTime(int *hour, int *minute, int *sec)

- **Parameter:**

  *hour: [output] The address of a variable used to receive the hour value of current time.

  *minute: [output] The address of a variable used to receive the minute value of current time.

  *sec: [output] The address of a variable used to receive the second value of current time.

- **Return:**

  None.

5.1.21 SetTime

- **Description:**
  Use this function to modify the time of real time clock.

- **Syntax:**
  int SetTime(int hour, int minute, int sec)

- **Parameter:**
  hour: [input] The hour value set to real time clock.
  minute: [input] The minute value set to real time clock.
  sec: [input] The second value set to real time clock.

- **Return:**
  _NO_ERR: OK
  _SET_TIME_ERROR: The input value of hour, minute or sec is invalid.

5.1.22 GetDate

- **Description:**
  Use this function to get the current date from real time clock.

- **Syntax:**
  void GetDate(int *year, int *month, int *day)

- **Parameter:**
  *year: [output] The address of a variable used to receive the year value of current date.
  *month: [output] The address of a variable used to receive the month value of current date.
  *day: [output] The address of a variable used to receive the day value of current date.

- **Return:**
  None.

5.1.23 SetDate

- **Description:**
  Use this function to modify the date of real time clock.

- **Syntax:**
  int SetDate(int year, int month, int day)

- **Parameter:**
  year: [input] The year value set to real time clock.
  month: [input] The month value set to real time clock.
  day: [input] The day value set to real time clock.

- **Return:**
  _NO_ERR: OK
  _SET_DATE_ERROR: The input value of year, month or day is invalid.

5.1.24 GetWeekDay

- **Description:**
  Use this function to obtain what day is today.

- **Syntax:**
  int GetWeekDay(void)

- **Parameter:**
  None.

- **Return:**

Return Code	Meaning
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

Table 5.2 Relation Between Return Code and Day of Week

5.1.25 ReadNVRAM

- **Description:**
  Use this function to get one-byte data of NVRAM.

- **Syntax:**
  int ReadNVRAM(int Address)

- **Parameter:**
  Address: [input] The NVRAM address where users will read the data. The range of this parameter is from 0 to 30.

- **Return:**
  _ACCESS_NVRAM_FAILE: The address of NVRAM is invalid.
  Others: The value obtained from NVRAM. The range of return value is from 0 to 255.

5.1.26 WriteNVRAM

- **Description:**
  Use this function to write one-byte data to specified address of NVRAM. If system has no power, the data stored in NVRAM will not disappear.

- **Syntax:**
  int WriteNVRAM(int Address, int data)

- **Parameter:**
  Address: [input] The NVRAM address where users will write the data. The range of this parameter is from 0 to 30.
  data: [input] The data written to NVRAM. The range of this parameter is from 0 to 255. If value is over 255, only low byte of data will be written to NVRAM.

- **Return:**
  _NO_ERR: OK.
  _ACCESS_NVRAM_FAILE: The address of NVRAM is invalid.

5.1.27 GetTimeTicks100us

- **Description:**
  Read I-8120W time ticks by using this function. When the firmware starts, I-8120W time ticks are counted. Reset the firmware will clean the accumulated counters of this value. If the accumulated counters are over the value 0xFFFFFFFF, the counters also reset to 0.

- **Syntax:**
  unsigned long GetTimeTicks100us(void)

- **Parameter:**
  None.

- **Return:**
  The time ticks numbers when firmware started. The unit is 0.1 ms.

5.1.28 GetTimeTicks

- **Description:**
  Call this function to read I-8120W time ticks. When I-8120W has power, the time ticks are counted. This function can't be called in interrupt service routine. Reset the operation system of I-8120W will clean the accumulated counters of this value. If the accumulated counters are over the value 0xFFFFFFFF, the counters are also reset to 0.

- **Syntax:**
  long GetTimeTicks(void)

- **Parameter:**
  None

- **Return:**
  The time ticks numbers. The unit is 1 ms.

5.1.29 DelayMs

- **Description:**

  Use this function to pending the procedure of user-defined firmware. Because of watchdog mechanism, users can't delay for a long time. The I-8120W watchdog timer is set to 800 ms. It is recommend that if users want to delay the procedure of user-defined firmware more than 500 ms. The the function RefreshWDT must be applied to avoid the watchdog timeout. This function is not allowed to put into interrupt service routine.

- **Syntax:**

  void DelayMs(unsigned int DelayTime_ms)

- **Parameter:**

  DelayTime_ms: [input] The delay time of procedure. The unit is 1 ms.

- **Return:**

  None

5.1.30 CM100_InstallUserTimer

- **Description:**

  This function can allow users to use timer interrupt. When users put their timer interrupt service routine in this function, this interrupt service routine will be executed every millisecond. Be careful that too much program in the interrupt service routine will disturb the normal procedure of user-defined firmware.

- **Syntax:**

  void CM100_InstallUserTimer(void (*Fun)(void))

- **Parameter:**

  (*Fun)(void): [input] The pointer which points a function with format "void XXX(void)". The XXX is the name of a function.

- **Return:**

  None

5.1.31 T_StopWatchXXX series functions

- **Description:**

  Call this function to use a stopwatch. There are 4 functions for stopwatch operation. When users want to start a stopwatch, the function T_StopWatchStart must be applied. Then, users can use the function T_StopWatchGetTime to obtain the current time counts of this stopwatch. If users need to disable the time counter, use the function T_StopWatchPause to achieve this purpose. Call the function T_StopWatchContinue to enable this timer counter again. If users want to use more than one stopwatch, just input the different variable of structure STOPWATCH into these 4 functions. One structure variable will be mapped to one stopwatch. The time unit of these 4 functions and the members of STOPWATCH structure are millisecond.

- **Syntax:**

  void T_StopWatchStart(STOPWATCH *sw)
  unsigned long T_StopWatchGetTime(STOPWATCH *sw)
  void T_StopWatchPause(STOPWATCH *sw)
  void T_StopWatchContinue(STOPWATCH *sw)

- **Parameter:**

  *sw: [output] The address of a STOPWATCH structure variable applied to describe the stopwatch. The member of STOPWATCH structure is shown as following:

```
typedef struct {
        unsigned long ulStart;
        unsigned long ulPauseTime;
        unsigned int uMode;
}STOPWATCH;
```

  Parameter ulStart obtains the start time of stopwatch. Parameter ulPauseTime will return the last pause time of stopwatch. Parameter uMode returns the status of the stopwatch. If uMode is 0, it means that the stopwatch pauses. If uMode is 1, the stopwatch is running.

- **Return:**

  The return code of T_StopWatchGetTime() is the current time counts after the stopwatch started.

5.1.32 T_CountDownTimerXXX series functions

- **Description:**

  Call this function to use a countdown timer. There are 5 functions for countdown timer operation. When users want to start a countdown timer, the function T_CountDownTimerStart must be applied. Then, If users need to disable the countdown timer, use the function T_CountDownTimerPause to achieve this purpose. Call the function T_CountDownTimerContinue to enable this countdown timer again. Users can use the function T_CountDownTimerIsTimeUp to check if the countdown timer is timeout or not. Or, use the function T_CountDownTimerGetTimeLeft to obtain the rest time of countdown timer. If users want to use more than one countdown timer, just input the different variable of structure COUNTDOWNTIMER into these 5 functions. One structure variable will be mapped to one countdown timer. The time unit of these 5 functions and the members of COUNTDOWNTIMER structure are millisecond.

- **Syntax:**

```
void T_CountDownTimerStart(COUNTDOWNTIMER *cdt,
                           unsigned long timems)
void T_CountDownTimerPause(COUNTDOWNTIMER *cdt)
void T_CountDownTimerContinue(COUNTDOWNTIMER *cdt)
int T_CountDownTimerIsTimeUp(COUNTDOWNTIMER *cdt)
unsigned long T_CountDownTimerGetTimeLeft(
                                    COUNTDOWNTIMER *cdt)
```

- **Parameter:**

  timems: [input] The time interval which indicates that how much time the countdown timer will countdown.

  *cdt: [output] The address of a COUNTDOWNTIMER structure variable used to describe the countdown timer. The member of COUNTDOWNTIMER structure is shown as following:

```
typedef struct {
        unsigned long ulTime;
        unsigned long ulStartTime;
        unsigned long ulPauseTime;
        unsigned int uMode;
} COUNTDOWNTIMER;
```

Using parameter ulTime will get time interval of countdown timer. Parameter ulStartTime returns the start time of countdown timer. Parameter ulPauseTime can obtain the last pause time of countdown timer. Parameter uMode returns the status of the countdown timer. If uMode is 0, it means that the countdown timer pauses. If uMode is 1, the countdown timer is running.

- **Return:**

The return code of T_CountDownTimerIsTimeUp() is _NO_ERR or _COUNT_DOWN_TIMER_TIME_UP. If the countdown timer is timeout, the return code is _COUNT_DOWN_TIMER_TIME_UP. If not, the return code is _NO_ERR. The return code of T_CountDownTimerGetTimeLeft() is the rest time of the countdown timer.

5.1.33 EEPROMReadByte

- **Description:**

Use this function to read the data of the specified address of EEPROM.

- **Syntax:**

int EEPROMReadByte(unsigned int Block, unsigned int Address,
unsigned char *Data)

- **Parameter:**

Block: [input] The EEPROM block No.. The range is from 0 to 6.

Address: [input] The EEPROM address where users will read the data. Each block has 256 bytes. Therefore, the range of this parameter is from 0 to 255.

*data: [output] The address of a variable used to obtain the data of specified address of EEPROM

- **Return:**

_NO_ERR: OK.

_EEPROM_OVER_RANGE: The block No. is over 6, or the address is over 256.

5.1.34 EEPROMReadMultiByte

- **Description:**
  Use this function to read some data from EEPROM.

- **Syntax:**
  int EEPROMReadMultiByte(unsigned int Block, unsigned int Address, char *Data, unsigned int DataNum)

- **Parameter:**
  Block: [input] The EEPROM block No.. The range is from 0 to 6.
  Address: [input] The start EEPROM address where users will write the data. Each block has 256 bytes. Therefore, the range of this parameter is from 0 to 255.
  *data: [output] The start address of a byte array used to receive the data from EEPROM
  DataNum: [input] The parameter indicates that how many data users want to obtain.

- **Return:**
  _NO_ERR: OK.
  _EEPROM_OVER_RANGE: The block No. is over 6, or the address is over 256. Or the specified range of reading data is over the block 6 and address 255.

5.1.35 EEPROMWriteByte

- **Description:**

  Use this function to write the data to specified address of EEPROM. If system has no power, the data stored in EEPROM will not disappear.

- **Syntax:**

  int EEPROMWriteByte(unsigned int Block, unsigned int Address, unsigned char Data)

- **Parameter:**

  Block: [input] The EEPROM block No.. The range is from 0 to 6.

  Address: [input] The EEPROM address where users will write the data. Each block has 256 bytes. Therefore, the range of this parameter is from 0 to 255.

  data: [input] The data written to EEPROM

- **Return:**

  _NO_ERR: OK.

  _EEPROM_ACCESS_ERROR: Can't write data to specified EEPROM address. The EEPROM may be damaged.

  _EEPROM_OVER_RANGE: The block No. is over 6, or the address is over 256.

5.1.36 EEPROMWriteMultiByte

- **Description:**

  Use this function to write some data to specified address of EEPROM. If system has no power, the data stored in EEPROM will not disappear.

- **Syntax:**

  int EEPROMWriteMultiByte(unsigned int Block, unsigned int Address, char *Data, unsigned int DataNum)

- **Parameter:**

  Block: [input] The EEPROM block No.. The range is from 0 to 6.

  Address: [input] The EEPROM address where users will write the data. Each block has 256 bytes. Therefore, the range of this parameter is from 0 to 255.

  *data: [output] The start address of a byte array used to store the data written to EEPROM.

  DataNum: [input] The parameter indicates that how many data users want to write.

- **Return:**

  _NO_ERR: OK.

  _EEPROM_ACCESS_ERROR: Can't write data to specified EEPROM address. The EEPROM may be damaged.

  _EEPROM_OVER_RANGE: The block No. is over 6, or the address is over 256. Or the specified range of writing data is over the block 6 and address 255.

5.1.37 UserCANIrqFunc *<must be called once>*

- **Description:**

  This is a callback function, and must be call once in user-defined firmware. When the firmware library receives an interrupt signal from CAN controller, this function will pass the interrupt indicator of CAN controller. The interrupt indicator shows what kinds of CAN controller interrupt are active. Therefore, users only need to design their interrupt routine according to deal with the different interrupt indicators. It is not allowed to put an infinite loop in to this function, and users must keep the program of this function as short as possible.

- **Syntax:**

  void UserCANIrqFunc(unsigned char INTT)

- **Parameter:**

  INTT: [input] The interrupt indicator from CAN controller. The meanings of indicators are shown as following.

Indicator (Hex)	Meaning
0x01	Receive a message successfully
0x02	Transmit a message successfully
0x04	Error warring
0x08	Data Overrun
0x10	CAN controller wake-up
0x20	Bus Passive
0x40	Arbitration Lost
0x80	Bus Error

Table 5.3 CAN Interrupt Indicator Description

- **Return:**

  None

5.1.38 SJA1000HardwareReset

- **Description:**
  Reset the CAN controller by reset the pin of SJA1000. After calling this function, users must configure the baud and message mask of CAN controller. Then, use the function EnableSJA1000 to activate the SJA1000 to send and receive CAN messages.

- **Syntax:**
  void SJA1000HardwareReset(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.39 SetCANBaud

- **Description:**
  Set the CAN baud of CAN controller.

- **Syntax:**
  int SetCANBaud(unsigned long Baud, char BT0, char BT1)

- **Parameter:**
  Baud: [input] The baud of CAN controller. There are 12 kinds of supported baud. They are 5K, 10K, 20K, 25K, 50K, 100K, 125K, 200K, 250K, 500K, 800K, 1M bps. If these bauds can not satisfy, set this parameter 0 and define the BT0 and BT1 of SJA1000.
  BT0: [input] User-defined baud.
  BT1: [input] User-defined baud. For the more information about how to use BT0 and BT1, please refer to the data sheet of SJA1000.

- **Return:**
  _NO_ERR: OK.
  _CAN_CHIP_SOFT_RESET_ERR: SJA1000 can't be reset by software. The CAN controller may be damaged.

5.1.40 GetCANBaud

- **Description:**
  Get the current CAN baud of CAN controller.

- **Syntax:**
  void GetCANBaud(unsigned long *Baud, char *BT0, char *BT1)

- **Parameter:**
  *Baud: [output] The address of a variable used to obtain the baud of CAN controller. If this parameter is 0, the BT0 and BT1 are useful.
  *BT0: [output] The address of a variable used to get the BT0 value obtained from SJA1000.
  *BT1: [output] The address of a variable used to get the BT1 value obtained from SJA1000. For more information about how to use BT0 and BT1, please refer to the data sheet of SJA1000.

- **Return:**
  None

5.1.41 SetCANMask

- **Description:**
  Set the message mask of CAN controller.

- **Syntax:**
  int SetCANMask(long AccCode, long AccMask)

- **Parameter:**
  AccCode: [input] Acceptance code of CAN controller
  AccMask: [input] Acceptance mask of CAN controller.
  The AccCode is used for deciding what kind of ID the CAN controller will accept. The AccMask is used for deciding which bit of ID will need to check with AccCode. If the bit of AccMask is set to 0, it means that the bit in the same position of ID need to be checked, and that ID bit value needs to match the bit of AccCode in the same position.

AccCode and AccMask	Bit Position	Filter Target
high byte of the high word	bit7~bit0	bit10 ~ bit3 of ID
low byte of the high word	bit7~bit5	bit2 ~ bit0 of ID
low byte of the high word	bit4	RTR
low byte of the high word	bit3~bit0	no use
high byte of the low word	bit7~bit0	bit7 ~ bit0 of 1st byte data
low byte of the low word	bit7~bit0	bit7 ~ bit0 of 2nd byte data

Table 5.3 AccCode and AccMask Definition For 11-bit ID

AccCode and AccMask	Bit Position	Filter Target
high byte of the high word	bit7~bit0	bit28~ bit21 of ID
low byte of the high word	bit7~bit0	bit20 ~ bit13 of ID
high byte of the low word	bit7~bit0	bit12 ~ bit5 of ID
low byte of the low word	bit7~bit3	bit4 ~ bit0 of ID
low byte of the low word	bit2	RTR
low byte of the low word	bit1~bit0	no use

Table 5.4 AccCode and AccMask Definition For 29-bit ID

For example (In 29 bit ID message):

AccCode	:	00h	00h	00h	A0h	
AccMask	:	FFh	FFh	FFh	1Fh	
ID bit		bit28~bit21	bit20~bit13	bit12~bit5	bit4~bit0	
ID Value	:	xxxx xxxx	xxxx xxxx	xxxx xxxx	101x x	will be accepted

(Note: The mark "x" means don't care. And the mark "h" behind the value means hex format.)

- **Return:**
  _NO_ERR: OK.
  _CAN_CHIP_SOFT_RESET_ERR: SJA1000 can't be reset by software. The CAN controller may be damaged.

5.1.42 GetCANMask

- **Description:**
  Get the current message mask status of CAN controller.

- **Syntax:**
  void GetCANMask(long *AccCode, long *AccMask)

- **Parameter:**
  * AccCode: [output] The address of a variable used to obtain the acceptance code of SJA1000.
  * AccMask: [output] The address of a variable used to obtain the acceptance mask of SJA1000.

- **Return:**
  None

5.1.43 CANConfig

- **Description:**

  Configure the baud, message filter of CAN controller. After calling this function, users need to call the function EnableSJA1000 to active CAN controller, SJA1000.

- **Syntax:**

  int CANConfig(unsigned long Baud, char BT0, char BT1, long AccMask, long AccCode)

- **Parameter:**

  Baud: [input] The baud of CAN controller.

  BT0: [input] User-defined baud.

  BT1: [input] User-defined baud.

  AccCode: [input] Acceptance code of CAN controller.

  AccMask: [input] Acceptance mask of CAN controller.

  For the more information about these parameters, please refer to the section 3.4.49 and 3.4.51.

- **Return:**

  _NO_ERR: OK.

  _CAN_CHIP_SOFT_RESET_ERR: SJA1000 can't be reset by software. The CAN controller may be damaged.

5.1.44 EnableSJA1000

- **Description:**

  Use this function to activate SJA1000. Afterwards, users can send/ receive CAN messages by other functions.

- **Syntax:**

  void EnableSJA1000(void)

- **Parameter:**

  None

- **Return:**

  None

5.1.45 DisableSJA1000

- **Description:**
  Call the function DisableSJA1000 to stop the functions of transmission CAN messages, reception CAN messages and interrupt.

- **Syntax:**
  void DisableSJA1000(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.46 GetCANStatus

- **Description:**
  Obtain the status register of SJA1000 by using this function.

- **Syntax:**
  int GetCANStatus(void)

- **Parameter:**
  None

- **Return:**
  The return code is the value of status register of SJA1000. Its meanings is described below.

Bit NO.	Description
7 (MSB)	Bus status. 1 for bus off, 0 for bus on.
6	Error status. 1 for at least one error, 0 for OK.
5	SJA1000 Transmit status. 1 for transmitting, 0 for idle.
4	SJA1000Receive status. 1 for receiving, 0 for idle.
3	SJA1000 Transmit complete status. 1 for complete, 0 for incomplete.
2	SJA1000 Transmit buffer status. 1 for released, 0 for locked
1	Data overrun status. 1 for SJA1000 reception buffer overrun, 0 for OK.
0 (LSB)	Receive buffer status. 1 for at least one message stored in the SJA1000 reception buffer, 0 for empty.

Table 5.5 The Description of Status Register of SJA1000

5.1.47 ClearDataOverrunStatus

- **Description:**
  When the data overrun status is obtained by using the function GetCANStatus, call this function to clear this status.

- **Syntax:**
  void ClearDataOverrunStatus(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.48 SendCANMsg

- **Description:**
  Send a CAN message to software transmission buffer. When the CAN bus is idle, this CAN message will be send to CAN network.

- **Syntax:**
  int SendCANMsg(char Mode, unsigned long MsgID, char RTR,
  char DataLen, char *Data)

- **Parameter:**
  Mode: [input] 0 for 11-bit message ID, 1 for 29-bit message ID.
  MsgID: [input] CAN message ID.
  RTR: [input] 0 for remote-transmit-request format is not used, 1 for remote-transmit-request is used.
  DataLen: [input] Data length of a transmitted CAN message. The maximum value is 8.
  *Data: [input] The start address of a buffer used to store the transmitted data of a CAN message.

- **Return:**
  _NO_ERR: OK.
  _SOFT_BUF_FULL: Transmission software buffer is full. Users need to transmit CAN message later. Or, use the function ClearTxSoftBuffer to clear the transmission buffer.

5.1.49 ClearTxSoftBuffer

- **Description:**
  Call this function to clear the transmission software buffer of CAN messages.

- **Syntax:**
  void ClearTxSoftBuffer(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.50 GetCANMsg

- **Description:**
  Obtain a received CAN message from the software buffer.

- **Syntax:**
  int GetCANMsg(char *Mode, unsigned long *MsgID, char *RTR,
  char *DataLen, char *Data, unsigned long *UpperTime,
  unsigned long *LowerTime)

- **Parameter:**
  *Mode: [output] The address of a variable used to get the mode of a CAN message. If value is 0, the received CAN message is with 11-bit ID. The 29-bit ID of a CAN message will have value 1.
  *MsgID: [output] The address of a variable used to get the CAN message ID.
  *RTR: [output] The address of a variable used to obtain the status of this CAN message. 0 for remote-transmit-request format is not used, 1 for remote- transmit-request is used.
  *DataLen: [output] The address of a variable used to get the data length of a CAN message. The range of this value is from 0 to 8.
  *Data: [output] The start address of a buffer used to get the data of a CAN message. Users need to put an 8-byte element array in this filed.
  *UpperTime: [output] The address of a variable used to obtain the higher double-word of time stamp of a CAN message.
  *LowerTime: [output] The address of a variable used to obtain the lower double-word of time stamp of a CAN message. The unit for UpperTime and LowerTime is 0.1ms.

- **Return:**
  _NO_ERR: OK.
  _RX_SOFT_BUF_EMPTY: The reception software buffer of CAN message is full. Users need to use ClearRxSoftBuffer() to clear this status when this return code is got.
  _SOFT_BUF_FULL: Reception software buffer is full. Users need to use function ClearRxSoftBuffer() to clear the CAN transmission buffer.

5.1.51 ClearRxSoftBuffer

- **Description:**
  Call this function to clear the reception software buffer of CAN messages.

- **Syntax:**
  void ClearRxSoftBuffer(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.52 RxMsgCount

- **Description:**
  Call this function to know how many available CAN messages stored in the reception software buffer.

- **Syntax:**
  int RxMsgCount(void)

- **Parameter:**
  None

- **Return:**
  The return code is the numbers of CAN messages stored in reception software buffer.

5.1.53 AddCyclicTxMsg

- **Description:**

  Add a cyclic transmission message into the cyclic transmission engine. Afterwards, uses can enable or disable this cyclic transmission messages by using the function EnableCyclicTxMsg and the function DelectCyclicTxMsg. Maximum 5 set of cyclic transmission messages can be applied. After adding a cyclic transmission message, the handle for this message will be returned. The less value of handle indicates the higher priority of this cyclic transmission message.

- **Syntax:**

  int AddCyclicTxMsg(char Mode, unsigned long MsgID, char RTR,
  char DataLen, char *Data,
  unsigned long TimePeriod, unsigned char *Handle)

- **Parameter:**

  Mode: [input] 0 for 11-bit message ID, 1 for 29-bit message ID.

  MsgID: [input] CAN message ID.

  RTR: [input] Set the remote-transmit-request is used or not. 0 is for useless, 1 is for useful.

  DataLen: [input] CAN message data length. The maximum value is 8.

  *Data: [input] The start address of the data buffer of a CAN message. The maximum space of *Data is 8 bytes.

  TimePeriod: [input] The time period of cyclic transmission. This parameter is formatted by 0.1ms. The minimum value is 5.

  TransmitTimes: [input] The numbers of CAN messages will be transmitted. After the I-8120W transmit all of the CAN messages which users decide the numbers of by using this parameter, I-8120W will disable this cyclic transmission message automatically. Users can enable this cyclic transmission message to require the I-8120W to send these CAN messages again by using the function EnableCyclicTxMsg. If this parameter is set to 0, the I-8120W will send CAN message cyclically and continuously after users enable this cyclic transmission message.

  *Handle: [output] The address of a variable used to get the handle of a cyclic transmission. When users want to enable or disable the

specified cyclic transmission, this value must be needed.

- **Return:**
  _NO_ERR: OK
  _CYCLIC_CONFIG_ERR: The cyclic transmission messages are over 5 messages or the time period is less than 0.5ms.

5.1.54 DeleteCyclicTxMsg

- **Description:**
  Remove a cyclic transmission message which is added by the function AddCyclicTxMsg before.

- **Syntax:**
  int DeleteCyclicTxMsg(unsigned char Handle)

- **Parameter:**
  Handle: [input] The handle of the cyclic transmission message which is obtained by the function AddCyclicTxMsg.

- **Return:**
  _NO_ERR: OK
  _CYCLIC_HANDLE_ERR: The handle value can't be found in the cyclic transmission engine.

5.1.55 EnableCyclicTxMsg

- **Description:**

  Enable a cyclic transmission message which is added by the function AddCyclicTxMsg before. After enable the specified cyclic transmission message, I-8120W will transmit the specified CAN message by configured time period.

- **Syntax:**

  int EnableCyclicTxMsg(unsigned char Handle)

- **Parameter:**

  Handle: [input] The handle of cyclic transmission message which is obtained by the function AddCyclicTxMsg.

- **Return:**

  _NO_ERR: OK

  _CYCLIC_HANDLE_ERR: The handle value can't be found in the cyclic transmission engine.

5.1.56 DisableCyclicTxMsg

- **Description:**

  Disable a cyclic transmission message which is enabled by the function EnableCyclicTxMsg before.

- **Syntax:**

  int DisableCyclicTxMsg(unsigned char Handle)

- **Parameter:**

  Handle: [input] The handle of cyclic transmission message which is obtained by the function AddCyclicTxMsg.

- **Return:**

  _NO_ERR: OK

  _CYCLIC_HANDLE_ERR: The handle value can't be found in the cyclic transmission engine.

5.1.57 ResetCyclicTxBuf

- **Description:**
  Clear the software buffer of cyclic transmission engine. After calling this function, all of the transmitted cyclic messages stop the procedure, and all of cyclic messages are removed from the cyclic transmission engine.

- **Syntax:**
  void ResetCyclicTxBuf(void)

- **Parameter:**
  None

- **Return:**None

5.1.58 SystemHardwareReset

- **Description:**
  Use this function to reset all hardware of I-8120W included 186 CPU.

- **Syntax:**
  void SystemHardwareReset(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.59 SystemInit

- **Description:**
  Use this function to initiate the DPRAM, LEDs, cyclic transmission engine, CAN transmission software buffer, and CAN controller.

- **Syntax:**
  void SystemInit(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.60 GetLibVer

- **Description:**
  Get the version of the firmware library.

- **Syntax:**
  int GetLibVer(void)

- **Parameter:**
  None

- **Return:**
  The return code is the version of the firmware library. For example: If 100(hex) is return, it means driver version is 1.00.

5.1.61 RefreshWDT

- **Description:**
  Call this function to refresh the watchdog of I-8120W. When users design the user-defined firmware, this function must be called where the users' procedure may have a processed period more than 500ms. If the function RefreshWDT is not called in 800ms, the 186 CPU of I-8120W will be reset.

- **Syntax:**
  void RefreshWDT(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.62 UserInitFunc *<must be called once>*

- **Description:**
  When users design the user-defined firmware, this callback function must be called once. Users can put some procedures into this function. These procedures are those which will be executed only one time in user-defined firmware. When I-8120W boots up, the firmware library will call this callback function once.

- **Syntax:**
  void UserInitFunc(void)

- **Parameter:**
  None

- **Return:**
  None

5.1.63 *UserLoopFunc* *<must be called once>*

- **Description:**

  When users design the user-defined firmware, this callback function must be called as soon as possible. Users can put their main procedures into this function. Then, the main procedure will be executed in every period of time. The time period is correlated with the complexity of users' main procedure. When I-8120W boots up, the firmware library will call the function UserInitFunc once and then call the function UserLoopFunc in every period of time until I-8120W is turned off. It is not allowed to put a infinite loop in this function.

- **Syntax:**

  void UserLoopFunc(void)

- **Parameter:**

  None

- **Return:**

  None

5.2 Firmware Library Return Codes Troubleshooting

If default firmware is used, users do not need to read this section.

Return Code	Error ID	Troubleshooting
-19	_SET_TIME_ERROR	1. Check the time format of input parameters, and retry it again.
-18	_SET_DATE_ERROR	1. Check the date format of input parameters, and retry it again.
-9	_ACCESS_NVRAM_FAILE	1. Try it again. 2. Call your distributor to solve this problem.
0	_NO_ERR	OK
1	_COUNT_DOWN_TIMER_TIME_UP	1. The countdown timer started by users is timeout.
101	_CAN_CHIP_SOFT_RESET_ERR	1. Call the function SJA1000HardwareReset, and try it again. 2. Call your distributor to solve this problem
102	_CAN_CHIP_CONFIG_ERR	1. Check the parameters of baud, BT0, BT1, acceptance code, and acceptance mask, and try it again.
103	_RX_SOFT_BUF_EMPTY	1. Wait for a while and call the function again.
104	_SOFT_BUF_FULL	1. Use the function ClearTxSoftBuffer or the function ClearRxSoftBuffer to clear the status of buffer overflow. 2. Reduce the bus loading of CAN network.
105	_DPRAM_WRITE_ERR	1. Wait for a while and call the function again. 2. Call your distributor to solve this problem
106	_DPRAM_READ_ERR	1. Wait for a while and call the function again. 2. Call your distributor to solve this problem.
107	_DPRAM_OVER_RANGE	1. Check the address or space range of written DPRAM, and try it again.
108	_NO_DPRAM_CMD	1. Wait for a while and call the function again.

Return Code	Error ID	Troubleshooting
109	_CYCLIC_CONFIG_ERR	1. Check if users already use 5 the cyclic messages. 2. Set the parameters TimePeriod to more than5.
110	_CYCLIC_HANDLE_ERR	1. Check the parameter Handle, and try it again.
111	_EEPROM_OVER_RANGE	1. Check the address or space range of written EEPROM, and try it again.
112	_EEPROM_ACCESS_ERROR	1. Wait for a while and call the function again. 2. Call your distributor to solve this problem.

Table 5.6 Return Code Troubleshooting

Note: If users' problem can't be fixed after following the recommended methods. Please contact your distributor or email to service@icpdas.com to solve the problem.